Assoziative Systeme als Datenspeicher

(Text storage and associative memories)

Vom Fachbereich IV
– Mathematik, Naturwissenschaften, Wirtschaft & Informatik –
der Universität Hildesheim

zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)
genehmigte

D i s s e r t a t i o n

von
Fabian Thomas Rosenschein
aus Laatzen

2012

1. Berichterstatter: Prof. Dr. Hans-Joachim Bentz
2. Berichterstatter: Prof. Dr. Günther Palm

Datei generiert am:	16. Juli 2013,
eingereicht am:	20. Dezember 2012
Datum der mündlichen Prüfung:	23. Januar 2013

Danksagung

Mein besonderer Dank gilt Prof. H.-J. Bentz. In den fruchtbaren Diskussionen der langjährigen Zusammenarbeit ergaben sich die Ansätze zu den präsentierten Ideen; und seine Heranführung an die Prinzipien assoziativer Speicher empfand ich als unschätzbar wertvoll.

Erklärung

Hiermit versichere ich an Eides statt, dass ich diese Abhandlung selbständig und ohne unerlaubte Hilfe verfasst und die benutzten Hilsfsmittel vollständig angegeben habe.

Laatzen, den 20.12.2012

Abstract

Associative Memories (AM) constitute a special form of Artificial Neural Networks (ANN) with only one layer of binary synapses. They perform well at tasks like pattern matching, extraction and completion when trained with pairs of sparsely coded binary vectors representing sets of properties.

While the achievable memory efficiency considering some measure of information content has been shown to be high, the probability for correct retrieval of *all* memorized association pairs is quite low.

For this reason it is not easy to use AM as memories for any given text or a list of instructions. Consequently AM were not used for direct storage of patterns when the original text has to be recalled, or adjoined AM are utilized – like in the 'Assoziativmaschine' – in conjunction with special key sequences, when program 'text' is to be stored.

This thesis describes an alternative approach, where the intrinsic association sequence of some (program as well as normal) text is preprocessed in a reversible manner to avoid clashes for maximized memory usage. Targeted experiments were undertaken in varying several parameters until satisfactory results could be found.

Resultingly large amounts of text can be stored associatively and retrieved without recall errors. As an illustration the whole textual content of *Alice In Wonderland (160kB)* and other texts were successfully stored. This even holds when the original text contains repetitive parts.

An example of another method using *long sequences in small memories* is then shown, which allows storing simple sequences in a more compact way – i.e. to implement microcode-like behaviour in an associative manner.

Specialized programs were developed to run and visualize those experiments.

Possible applications can be found in operation of and controlling the Assoziativmaschine. Furthermore – because of the robustness of storage – the technique may prove beneficial in the field of archiving/digital preservation.

Zusammenfassung

Assoziativspeicher (AM) stellen eine besondere Form (künstlicher) Neuronaler Netze (KNN) mit einer Schicht dar. Sie eignen sich zur Anwendung bei Aufgaben wie Mustererkennung, -extraktion und -ergänzung, wobei Lernpaare von spärlich besetzten Binärvektoren, welche Ansammlungen von Merkmalen darstellen, verwendet werden.

Die aus „Lernmatrizen“ nach Steinbuch weiterentwickelten Matrizen haben nicht-lineares Verhalten ohne negative Gewichte, sind binär – und daher einfach zu konstruieren – und somit einfacher mit Texten (technisch: Zeichenfolgen) in Verbindung zu bringen.

Während bekannt ist, dass die erreichbare Speichereffizienz als gut angesehen werden kann, ist dagegen die Wahrscheinlichkeit des korrekten Wiederfindens *aller* eingespeicherten Assoziationsmuster gering.

Entsprechend wurde das Problem, *beliebige* Textdaten direkt in Assoziativspeichern aufzuheben, lange nicht angegangen.

In der „Assoziativmaschine“ [(1)] wurden bereits in Matrizen gespeicherte Programme, Zustandsvariablen und Weltwissen in einer Kombination solcher Speicher derart verschaltet, dass eine algorithmische Steuerung realisiert werden konnte. Eine besondere Rolle spielen hier „Sequenzen“, welche als Folgen einen geordneten Zugriff auf Daten ermöglichen.

Wir überprüfen jetzt unterschiedliche Ansätze und Aspekte dieser Aufgabe, Texte mit assoziativen Strukturen zu verarbeiten und zu speichern. Zur Durchführung und Veranschaulichung wurden spezielle Programme erstellt.

Letztlich ergibt sich die Möglichkeit, grössere Mengen von Text vollständig assoziativ zu speichern und fehlerlos wieder auszulesen, was am Beispiel des gesamten Textes von *Alice's Abenteuer Im Wunderland (160kB)* demonstriert wird. Das dargestellte Verfahren funktioniert auch dann, wenn der Originaltext Wiederholungen enthält.

Unter anderem lässt sich als einzige Version einer Zeichenfolge eine schnell durchsuchbare Kodierung aufbewahren.

Schliesslich wird ein weiteres Verfahren gezeigt, welches *lange Sequenzen in kleinen Matrizen* nutzt. Hier können einfache Abfolgen in kompakter Weise abgelegt werden, zum Beispiel zur assoziativen Implementierung von Microcode-artigem Verhalten.

Mögliche Anwendungen finden sich bei Betrieb und Steuerung der Assoziativmaschine. Weiterhin können Vorzüge dieser Technik – wie die Robustheit des Speichers – im Bereich der Langzeitdatenspeicherung/Archivierung genutzt werden.

[(1)] beschrieben in [Dierks 05] aus unserer Arbeitsgruppe

Inhaltsverzeichnis

1. Einleitung

Matrixspeicher als spezielle *künstliche neuronale Netze (KNN)* werden bisher wie diese allgemein verwendet, um Muster zu erkennen, vervollständigen oder extrahieren [(1)] .

In unserer Arbeitsgruppe [(2)] wurden als Datenmengen speziell dabei oft *Texte* verwendet, sei es

- zur fehlertoleranten Suche (vgl. [Bentz 06]),
- zur Klassifikation von Texten (vgl. [Frobese 09]) oder
- zur Zuordnung von Texten (vgl. [Na nhongkai 06]).

Zum Teil spielte hierbei die verwendete Kodierung der Texte nur eine untergeordnete Rolle [(3)] , einfache Paarkodierungen erwiesen sich oft als hinreichend.

Schon [Palm 82] behandelt Kodierungen ausführlich; [Hagström 96] untersucht Positions- und Tupel-Kodierungen besonders auf Spärlichkeit, [Kopold 97] beschäftigt sich mit ähnlichkeitserhaltenden Kodierungen.

Für speziellere Anwendungen wurden auch andere Kodierungen implementiert [Luba 01]. [Palm 82] und [Palm 13] erwähnen *bidirectional retrieval*, meinen hiermit jedoch eine erweiterte Autoassoziation; in [Sommer, Palm 99] finden sich dann Beschreibungen, Sequenzen durch Iteration zu *liefern*, nicht jedoch gezielt zu speichern.

In [Janotta 11],[3] sind dagegen systematisch alle Merkmalsextraktoren bestimmter Typen auf einzelnen Worten für einen bestimmten Anwendungszweck untersucht worden, allerdings ohne eine Umsetzung mit mehreren Assoziativspeichern zu realisieren.

Die hier und in anderen Projekt- und Studienarbeiten gemachten Erfahrungen, sowie Ergebnisse anderer Gruppen wie [Hawkins et al. 11] und [Fay 02] legen ebenfalls nahe, teilweise *mehrere* Matrixspeicher zu kombinieren.

A. Dierks hingegen verwendet die *Kombination von Matrixspeichern* zur Konstruktion von *Assoziativmaschinen*, wenn auch in einem anderen Sinne, typischerweise mit einer klaren Trennung von Aufgaben der verschiedenen Speicher;

---

(1) [Palm 80], [Palm 88] S.62

(2) Institut für Mathematik und Angewandte Informatik (IMAI) – Abteilung Diskrete Mathematik, Stochastik und Neuromathematik – am Fachbereich IV der Universität Hildesheim

(3) gemeint ist hier vordergründig nicht die Zeichenkodierung, sondern die Aufbereitung eines Textes in eine Menge von Merkmalen wie *„das fünfte Zeichen ist ein* <**A**> *“* oder *„ der Text enthält (irgendwo) die Zeichenfolge* <**SCH**> *“*

auch nutzt er zum Teil andere Lernregeln [4] als wir sie verwenden werden.

Wir nutzen nun die gewonnenen Erkenntnisse, indem Matrixspeicher verwendet werden – allerdings mit einer neuartigen Zielsetzung:

Während bisher die Extraktion von Merkmalen und nachfolgende Einschreibung in einen oder mehrere Matrixspeicher entweder *die einzelnen Merkmale* oder *die einzelnen Datensätze* als technisch unabhängig betrachtete – insbesondere die Reihenfolge der Einschreibung [5] dabei keine Rolle spielte – sich also insbesondere die ursprüngliche Speicherfolge nicht rekonstruieren liess, *so wollen wir nun diese ursprüngliche Folge immer auch rekonstruieren können.*

Die Assoziativspeicher dienen also endlich nicht allein der Merkmalsverarbeitung, sondern diese ist auch Mittel zum Zwecke der Speicherung beliebiger textueller Information und des späteren gezielten und identischen Abrufes.

Wir sehen hier auch eine Parallelität zum menschlichen Denken und Erinnern, welches zwar durchaus das Auswendiglernen eines Gedichtes erlaubt, aber – wie unsere Assoziativspeicher und Assoziativmaschinen – im Gegensatz zu einer Rechenanlage mit freiadressierbarem Speicher (RAM) dies nicht durch Zuordnung eines einzelnen Zeichens zu einem bestimmten Speicherplatz zu erledigen scheint.

Bei Dierks' Assoziativmaschine finden sich Ansätze hierzu, indem dort in der *Abfolgematrix* und allgemein zur *Repräsentation von Ordinalzahlen* spezielle Zählfolgen betrachtet werden [6] . Die abgelegten Folgen sind jedoch nicht beliebig, sondern werden in engen Grenzen von einem Simulationssystem generiert [7] .

Später findet sich dieses Konzept auch im sogenannten *Fortsetzungskern* [8] als Teil von anderen hierdurch geordneten Merkmalsfolgen.

Hierauf bauen wir im ersten Teil der Arbeit zunächst auf und verwenden später dann die bereits gegebenen Merkmale in den betrachteten Texten selbst zur *fortgesetzten Assoziation* [9] .

Speziell dadurch ergeben sich bestimmte Optionen, wobei hier nur folgendes angedeutet werden soll:

- Offensichtlich besteht keine Notwendigkeit, sich eine besondere Zählfolge

[4] vgl. [Dierks 05] S.68ff:„Daher erfährt die Matrix, die das Kurzzeitgedächtnis repräsentiert, vor jedem Eintrag [...] eine Löschung [...] in den zum Bezeichner gehörenden Matrixzeilen"

[5] so zumindest mit der L-Lernregel (bei Dierks [Dierks 05] S.16ff im sogenannten Langzeit-Gedächtnis)

[6] „Dazu wurden zwei Assoziativmatrizen eingesetzt: die erste assoziierte zu einer gegebenen Situation [...] die nachfolgende" ([Dierks 05] S.55)

[7] „erhielt jede Programmzeile als Zeilennummer eine selbstgenerierte Zufallszeichenkette" (ebd. S.56)

[8] in [Bentz, Dierks 12] Kap. 6

[9] ähnlich dem *frageweiter*-Befehl in [Dierks 05] S.96: „Das schafft die Möglichkeit, Assoziationsketten zu folgen"

„an anderer Stelle“ zu merken, wie bei Dierks in den dort notwendigerweise synchronisierten *Abfolge- und Befehls-Matrizen* [(10)] .

- Ausserdem verbleibt die Darstellung im Matrixspeicher näher am oder auch ganz im (natürlichen) Modellierungsraum der Textdaten selbst, was die Struktur der Zeichen-, Wort- und Satz-*Folgen* einer Verarbeitung auch auf anderen Ebenen zugänglich macht – zum Beispiel zum statistischen Abgleich von Semantik, vergleichbar mit Bayesschen Filtern.
- Denkbar ist auch die Ablaufsteuerung von Programmen oder Maschinen, und damit nicht zuletzt auch die Optimierung der in der Textspeicherung selbst verwendeten Informationsflüsse – mit den speziellen Eigenschaften von Assoziativmatrizen: Fehlertoleranz, Adaptivität und Geschwindigkeit.

Natürlich sind dabei auch besondere Schwierigkeiten zu überwinden – zum Beispiel das Verharren in Zyklen, welches durch die „ähnlichen Datenbereiche“ in den verarbeiteten Texten auftreten kann.

Insgesamt wird aber gezeigt werden, dass eine Speicherung von Texten allein mit immanenten Merkmalen regelmässig möglich ist.

Wir untersuchen verschiedene Darstellungen von Texten, welche zum Teil sich ergänzend zur Rekonstruktion eines beliebigen Datenmusters genutzt werden können.

Die gefundene transformierte Darstellung kann als einzige gespeicherte Version einer Zeichenfolge auch schnell durchsucht werden.

[(10)] vgl. Abb. 41 „Abfolge von Programmschritten“ in [Dierks 05] S.57

2. Grundlagen

Wir beschreiben kurz das mathematische Modell des *Assoziativspeichers*.

Anschliessend geben wir noch Beispiele für die von uns verwendeten Begriffe *Assoziation* und *Assoziationskette* .

(Gelegentlich werden hier bereits Begriffe – wie zum Beispiel *„Lernregel L“* – verdeutlichend und um einen Überblick zu ermöglichen verwendet, welche erst im gleich folgenden Abschnitt 3.2 definiert sind.)

2.1. Lernmatrizen und Assoziativspeicher

Wie eingangs geschildert, ist **Lernmatrix** eine traditionelle Bezeichnung für das von uns verwendete Speichermodell, anfangs aus analogen Teilen und keinesfalls binär operierend.

Für einen Überblick der Geschichte der daraus entwickelten Matrixspeicher siehe [Dierks 05] S.23 ff; [Palm 82] sowie das umfangreiche [Bentz, Dierks 12].

Im Laufe der Zeit fanden sich – zum Teil nur dem Namen nach ähnliche Systeme beschreibend – dann Begriffe wie „Associative Computing“ [(1)] , welche auch dem Prinzip der Daten-Gesteuertheit folgen, um Datenbank-ähnliche Abfragen auf Tupeln aufgezählter Mengen zu implementieren.

Wenn *wir* in einer *Assoziation t* das Zusammentreffen eines Begriffes in einer Anfrage F mit einem Begriff in einer Antwort A beschreiben, so ist dieses Zusammentreffen – einer Korrelation $C > 0$ entsprechend – nur noch ein boolscher Term, formuliert über allen möglichen Kombinationen von Merkmalen in den denkbaren „Fragen“ und „Antworten“ [(2)] .

Wegen der dementsprechend binären Grundelemente unserer Betrachtungen, verwenden wir regelmässig den eher technischen Term **Assoziativspeicher**, insbesondere auch bei der Merkmalsverarbeitung auf Textmengen.

Mit gleicher Bedeutung reden wir von diesen Speichermatrizen ebenso als *Assoziativmatrix* oder *Matrixspeicher* und wegen der dahinterstehenden mathematischen Struktur eben auch kurz (binäre) *Matrix*.

Die Verwendung des Begriffes **Speicher** findet ihre Rechtfertigung auch darin, dass zumeist die Werte direkt eingeschrieben und nicht wie bei anderen KNN

(1) [Potter 92]

(2) vgl. das Beispiel der „Einzelassoziation“ in Kap. 2.4

in vielen Iterationen „gelernt“ werden. Dennoch verwenden wir auch den Begriff **„lernen“**, um das Einschreiben einer Assoziation in den Speicher zu beschreiben.

Der Speicher ist dabei eine binärwertige Matrix im mathematischen Sinne, welche formal [(3)] einem einschichtigen KNN gleicht; jedoch wird zum Bestimmen der Ausgangswerte eine besondere, die sogenannte *Abfrageregel R* [(4)] verwendet.

2.2. Lernregeln, Merkmale und Kodierungen

Wenn man die Speicher- beziehungsweise Synapsenstruktur der von uns verwendeten Assoziativspeicher als einschichtige [(5)] binäre künstliche neuronale Netze auffasst, so stellt sich das Einspeichern von Texten als Einstellung von Synapsen-Gewichten dar.

Entsprechend den Bezeichnungen bei KNN sprechen wir auch von Lernen (hier von Assoziationen) durch *Anwenden von Lernregeln.*

In [Dierks 05] werden die Regeln L und K für das *Lang- und Kurzzeitgedächtnis der Assoziativmaschine* vorgestellt.

L ist dabei eine klassische (Hebb-artige) Vorschrift zur Änderung von Gewichten, gerade soweit wie sie sich für alle paarweisen Kombinationen von Einzelmerkmalen der Lernpaare (jeweils Frage F^t und Antwort A^t) im binären Falle zwangsläufig ergibt [(6)] .

K ist eine nicht-Hebb-artige Variante, welche dort gewährleistet, dass Assoziationen auch wieder aus dem Speicher gelöscht werden können. Diese werden wir anfangs nicht betrachten, da unser Augenmerk darauf gerichtet ist, eine (dauerhafte) Repräsentation eines spezifischen und sich nicht ändernden Textes zu finden.

2.3. Eigenschaften und Abgrenzung

Assoziativmatrizen (AMs) haben gegenüber „normalen“ Neuronalen Netzen (NNs) als eine spezielle Form besondere Eigenschaften:

- Datensätze (Assoziationen) lassen sich direkt einspeichern, im Gegensatz zu einer sonst üblichen „längeren“ Trainingsphase.

---

(3) [Rojas 93] S. 122

(4) s. Abschnitt 3.2

(5) In der Literatur wird dies verschiedentlich auch anders bezeichnet; so schreibt z.B. sogar [Heitland 94] (S.19): „die Netzstruktur weist zwei Schichten auf“ – wir verzichten auf eine genauere Diskussion, halten uns jedoch hier an [Rojas 93] S. 122

(6) s. Abschnitt 3.2

- Ein- und Ausgaben sind bereits Binärdaten, was sie zur Verarbeitung von Informationen in und für Computersysteme prädestiniert.
- Durch die Reduktion des Ergebnisvektors auf einem vorher festgelegten Schwellwertniveau oder (in dieser Arbeit zumeist) auf der Maximalschwelle der Netzantwort ergibt sich eine Unstetigkeit im Übertragungsverhalten; unter anderem können so auch Negationen ohne negative Gewichte modelliert werden – welche es ja bei der boolschen Wertemenge $\{0, 1\}$ bei den Gewichten nicht gibt.
- Zur Berechnung eines Outputs zu einem gegebenen Input werden wegen dieser Gewichte keine Multiplikationen, sondern nur eine technische Möglichkeit zum Zählen benötigt (alle Summanden sind 0 oder 1), wodurch einerseits andere Grössenordnungen von Schichten im Bereich mehrerer tausend Neuronen völlig unproblematisch simuliert werden können, und andererseits ist der Rechenprozess auch einfach sowohl horizontal als auch vertikal aufzuteilen (parallelisieren) [7] – der notwendige Abgleich zum Bilden des Gesamtschwellwertes ist nicht aufwendig.

Aus diesen Eigenschaften resultierend stehen die assoziativen Matrizen nicht in direkter Konkurrenz zu anderen technischen Methoden wie SVM und bayesschen Klassifikatoren oder auch schlichten Inhalts-adressierten Speichermodulen in Cache-Speichern zum schnellen Zugriff:

Weder werden die Datensätze als reellwertige Vektoren aufgefasst, noch ist ausschliesslich eine spezifische Sequenz von Ausprägungen bestimmend für das Resultat der einzelnen Berechnung. Und im Gegensatz zu Cache-Speichern kann die gespeicherte Datenmenge sehr gross werden; dies sowohl nach Anzahl der Datensätze, aber auch die Entsprechung einer einzelnen Adresse selbst kann mehr als 10000 Bits aufweisen.

Stattdessen werden bei assoziativer Modellierung Kombinationen von Attributen (Einzelmerkmalen) von Datensätzen in Binärvektoren als Repräsentanten [8] letztenendes von „Beobachtbarem“ verwendet, ohne dass eine feste Struktur der Daten gegeben sein muss.

(Insofern ähnelt dieser Ansatz auch dem Herangehen, welches sonst im Bereich der Programmierung als *aspektorientiert* [9] bezeichnet wird: Es werden Einzelmerkmale zunächst einmal als unabhängig modelliert [10] .)

Wegen der Grösse der Matrizen können dabei auch auf den ersten Blick nichtrelevante Merkmale mit verarbeitet werden – durch die inhärente Geschwindig-

[7] ersteres wurde, wenn auch aus anderer Notwendigkeit und nicht auf verteilten Rechenanlagen bereits im Abschnitt „Segmentierte Matrizen“ in [Hagström 96] (S.60 ff.) diskutiert

[8] Dierks nennt sie in [Bentz, Dierks 12] auch Eindrücke

[9] oder teilweise *Mixins*, oder „Feature-Oriented Programming (FOP)“

[10] „cross-cutting“

keit des Rechenprozesses und besondere Arten der Darstellung der Matrix [11] ist es nicht unbedingt notwendig, hier von Anfang an eine Reduktion auf notwendige Merkmale vorzunehmen.

Wir wollen Assoziativmatrizen jedoch jetzt als Speicher nutzen, also zumindest aus Nutzersicht nicht als Operatoren auf Merkmalsmengen, obwohl die Implementierung darauf beruht. Als Seiteneffekte erhalten wir möglicherweise auch einige der bereits in der Literatur beschriebenen Vorteile umsonst dazu:

- *Schnelles Suchen,*
- *Fehlertolerante Suche,*
- *Robustheit*

[11] „Modelle mit Zeigern und Feldern“ in [Hagström 96] S.33 ff. untersucht Speicher- und Rechenaufwand für reduzierte Darstellungen von Speichermatrizen

2.4. Besondere Begrifflichkeiten unserer Modelle: Assoziationen und Ketten

Assoziationen, Assoziationskette: Als **Assoziation** bezeichnen wir ein Paar $\tau_t = (F^t, A^t)$.

Die F und A können Begriffe, Objekte abstrakter oder auch konkreter Natur – wie Zahlen – oder Textstücke wie Zeilen oder Worte oder Ausschnitte davon *repräsentieren*.

Damit diese Repräsentationen von einem Assoziativspeicher sinnvoll verarbeitet werden können, handelt es sich tatsächlich um eine Konjunktion [12] von mindestens 2 einzelnen (atomaren) Merkmalen binärer Ausprägung.

Beispiel:

<**VR**> [13] : „Vertikal $\oplus$ halbRund“ $\rightarrow$ Mond,

<**RH**> : „halbRund $\oplus$ Horizontal“ $\rightarrow$ Gondel, Barke,

<**HW**> : „Horizontal $\oplus$ Wasser“ $\rightarrow$ Kanal, See,

<**VW**> : „Vertikal $\oplus$ Wasser“ $\rightarrow$ Regen.

In Lesereihenfolge könnten wir kurz feststellen:

„Im Mondlicht sehen wir die Gondel auf dem Kanal. Regen.“

Eine Abfolge von τ_t wäre also möglicherweise:

$$\tau_0 = (\ \text{<}\mathbf{VR}\text{>}\ ,\ \text{<}\mathbf{RH}\text{>}\ )\ ,$$
$$\tau_1 = (\ \text{<}\mathbf{RH}\text{>}\ ,\ \text{<}\mathbf{HW}\text{>}\ )\ ,$$
$$\tau_2 = (\ \text{<}\mathbf{HW}\text{>}\ ,\ \text{<}\mathbf{VW}\text{>}\ )\ .$$

Durch diese einzelnen Assoziationen, selbst wenn wir sie – denn wie oben angemerkt betrachtet ein Assoziativspeicher mit der Lernregel L alle Datensätze als natürlicherweise gleichberechtigt – in der Reihenfolge $2, 0, 1$ notieren würden, wäre doch die Sequenz der Hauptworte des angegebenen Satzes festgelegt.

Derartige überlappende 2-Tupel $\tau_t = \tau_0, \ldots, \tau_2$ mit $t > 0 : F^t = A^{t-1}$ bezeichnen wir als **(Assoziations-)Kette**.

---

[12] Wir halten uns hier an die Begrifflichkeiten der Merkmalsmengen/-vektoren. Da jedes Merkmal von anderen unterschieden werden können soll, identifizieren wir es später mit einer Ordinalzahl. Als Binärvektor geschrieben entspricht eine Menge von Ordinalzahlen dann den Stellen (Indizes) der 1-en, die Menge ist also eine Indexmenge.

Das *gemeinsame* Auftreten von Merkmalen wird durch die *Oder*-Verknüpfung ihrer *einzelnen Beschreibungen als Binärvektoren* symbolisiert. Dies lässt sich allerdings auch im disjunktiven Sinne verstehen, da eine volle Übereinstimmung nur bei gleichzeitigem Vorliegen *aller* Einzelmerkmale $\bigwedge M_i$ möglich ist; durch die Reduktion auf der nicht-konstanten Schwelle des jeweiligen Maximums wird bei den Assoziativspeichern diese Anforderung jedoch aufgeweicht.

[13] oder RV, eine „korrektere“ Notation wäre hier {**V,R**}

Einzelassoziation: Die beispielsweise durch τ_0 implizit festgelegten hier 4 Zusammentreffen

$$\{\ \texttt{V}\ \wedge \texttt{R}\ ,\quad \texttt{V}\ \wedge \texttt{H}\ ,\quad \texttt{R}\ \wedge \texttt{R}\ ,\quad \texttt{R}\ \wedge \texttt{H}\ \}$$

bezeichne ich im folgenden gelegentlich als *Einzelassoziationen*; sie entsprechen gerade einer gesetzten 1 im Matrixspeicher.

Symbole: Wir können leicht sehen, dass die Symbole beliebig wählbar sind:

$$\texttt{|C}\ \mapsto \texttt{C-}\ ,\qquad \texttt{C-}\ \mapsto \texttt{-˜}\ ,\qquad \texttt{-˜}\ \mapsto \texttt{|˜}\ ,$$

wobei nur

- `|` statt <**V**> ertikal,
- `C` statt halb <**R**> und,
- `-` statt <**H**> orizontal, sowie
- `˜` statt <**W**> asser

geschrieben ist, ist ebenso denkbar und erzählt dieselbe Geschichte.

Zeilenweise gelistet sehen wir die einzelnen Merkmale [14] :

t	F^t				A^t			
	C	-	|	˜	C	-	|	˜
0	1	0	1	0	1	1	0	0
1	1	1	0	0	0	1	0	1
2	0	1	0	1	0	0	1	1

Es ist jedoch festzuhalten: Hieraus lässt sich nicht verlässlich, also eindeutig, der bezeichnete Satz rekonstruieren: Weder ist der Betrachter enthalten, noch sind es die Artikel und Verben.

Auch die Wahl von Gondel $\leftrightarrow$ Barke ist beliebig.

Wir wollen daher nun Kodierungen betrachten, welche mehr am Buchstaben festhalten.

---

[14] und erhalten als Speichermatrix (0-en kurz als <.> notiert):

(a_{ij}^t)	R	H	V	W
R	1	1	.	1
H	.	1	1	1
V	1	1	.	.
W	.	.	1	1

Hieraus kann durch Anfragen „von links“ von zuerst <**RV**> nacheinander rekonstruiert werden:

$$\rightarrow \text{<}\mathbf{RH}\text{>} \rightarrow \text{<}\mathbf{HW}\text{>} \rightarrow \text{<}\mathbf{VW}\text{>} \rightarrow \text{<>}\ .$$

3. Assoziative Systeme als Datenspeicher

3.1. Beschreibung; Motivation

Wenn wir direkt wie in Abschnitt 2.4 vorgehen, um nun *Zeichenfolgen* über dem Alphabet

<**A**> ... <**Z**> $\oplus$ <**.**>

in einen Assoziativspeicher zu schreiben, indem wir als die (mindestens) zwei Grundmerkmale immer die benachbarten Buchstaben verwenden, erhalten wir für den Beispieltext

<**Im Mondlicht sehen wir die Gondel auf dem Kanal. Regen.**>

unter anderem die Assoziationen:

$\tau_0 = (\ \{\mathbf{I,m}\}\ ,\ \{\mathbf{m,}\textvisiblespace\}\ )\ ,$

$\vdots$

$\tau_4 = (\ \{\mathbf{o,n}\}\ ,\ \{\mathbf{n,d}\}\ )\ ,$

$\tau_5 = (\ \{\mathbf{n,d}\}\ ,\ \{\mathbf{d,l}\}\ )\ ,$

$\vdots$

$\tau_{28} = (\ \{\mathbf{o,n}\}\ ,\ \{\mathbf{n,d}\}\ )\ ,$

$\tau_{29} = (\ \{\mathbf{n,d}\}\ ,\ \{\mathbf{d,e}\}\ )\ ,$

$\vdots$

doch niemand kann einfach sinnvoll die Frage beantworten [1] :

„Welches 2-Buchstaben-Tupel folgt auf das {**n,d**} ?“

Als Aufgabe stellt sich also, zusätzliche Fingerzeige zu realisieren oder anderweitig von einer so einfachen Assoziationskette abzuweichen, um Texte *so* zu speichern, dass eine vollständige und eindeutige Rekonstruktion des Ursprungstextes aus den einzelnen Assoziationen möglich wird.

Zuerst jedoch wollen wir kurz in Gedanken einen Schritt zurücktreten und uns die mögliche Interaktion mit einem KNN einmal vor Augen führen.

---

[1] denn *das* {**n,d**} gibt es nicht, es sei denn, man zeigt es genauer mit dem Finger.

Exkurs: Geschichten erzählen...

Im diesem kurzen – informellen und anekdotischen – Abschnitt wird als Rekapitulation, Ausblick und Motivation noch einmal die bei Verwendung von Assoziativspeicheren typische Denkweise erläutert und was sie von den sonst oft üblichen Herangehensweisen an Probleme der Datenverarbeitung unterscheidet.

Stellen Sie sich bitte ein typisches (künstliches) Neuronales Netz vor. Bitten Sie es, Ihnen eine Geschichte zu erzählen. Nehmen wir an, Sie hätten es mit Börsenkursen trainiert. Was wird es erzählen? Wenn alle Eingänge zunächst auf normiert 0 liegen, erzählt es, was durch Integration einer langweiligen Woche in der Vergangenheit sich dann an Kursschwankungen ergab, zunächst eine Prognose für einen oder wenige Tage.

Hier hört das übliche KNN aber auch auf. Nicht, weil es nicht weiter könnte, sondern weil man es einfach normalerweise nicht fragt!

Diese Nicht-Idee stammt wohl aus den folgenden zwei Trug-Überlegungen:

- Wenn ich mehr wollte, als von historischen $H_{i-k+1,\ldots,i}$ auf den zukünftigen F_{i+1} zu schliessen, dann hätte ich doch was anderes gelernt!

  Und eng damit zusammenhängend:

- Ich will doch heute nicht wissen, ob ich *übermorgen* kaufen oder verkaufen soll; das weiss ich doch *morgen viel sicherer*, wo ich dann F_{i+1} auch schon kenne!

Und das Netz wird dann gut (trainiert) genannt, wenn es möglichst genau die ihm beigebrachte Geschichte erzählen kann, obwohl es später ja wohl niemals diese Geschichte erzählen sollen wird! In dem Moment, wo nämlich tatsächlich exakt dieselben Werte auftreten wie bereits in der Vergangenheit, kann auch jeder normale Mensch sagen, „wie die Geschichte [(2)] weitergeht“. Einmal davon abgesehen „erwartet“ man allerdings dann gerade oft, „dass *„die“* diesmal doch nicht wieder genauso dumm sind“; das Netz dagegen will hier Zyniker sein.

Dieses – dass man eigentlich keine „innovativen Ergebnisse“ erhalten möchte – erklärt wohl auch, warum bisherige reale Netze gar nicht mal so kompliziert sind! Nur ein Dutzend Schichten mit ebensovielen Neuronen sollen meistens ausreichen.

Die herkömmliche Betrachtungsweise motiviert das mit „mehr kann man nicht mehr trainieren“ oder „es würde nicht besser“.

Eine Ausnahme stellen hier die Hirnforschung und verwandte Gebiete dar, wo es zum Teil auch darum geht, Kreativität zu verstehen; ein einfacher Weg dazu ist, das Netz nahe an den Grenzen seiner Kapazität zu betreiben:

[(2)] egal ob es sich um einen Aktienkurs oder eine andere Entwicklung handelt; natürlich ist diese Darstellung extrem vereinfacht, da man oft nicht einmal weiss, welche Kenngrössen aus der Realität überhaupt passende Eingangswerte für ein KNN sein mögen

Gerade dann haben kleine Schwankungen plötzliche Auswirkungen. Da sich die Wirkungen in *konventionellen KNN* (also nicht *Assoziativspeichern* nach unserer Definition) im allgemeinen gemittelt linear-additiv ergeben, bedeuten diese immer eine Nutzung in der Nähe einer nicht im Sinne einer harmonischen Regelung sinnvollen Eskalation der Werte.

Typischerweise müssen die Werte ja aber sogar noch normiert werden, was diese Eskalation wieder ein wenig zu begrenzen scheint. Dennoch ist diese Dämpfung nicht gut steuerbar, weswegen diese Netze rückgekoppelt und in den Randbereichen dann vermutlich chaotisches Verhalten zeigen, was durch die beschriebene starke Auf- und Abskalierung bedingt wird. Phänomene wie Rundungsfehler überdecken diesen Effekt dann möglicherweise wieder, wenn die Eingaben nicht in einem wohldefinierten Bereich liegen.

Im Mittelbereich dagegen, wo ein Arbeitspunkt intuitiv liegen sollte, können dagegen auch grosse Schwankungen nichts bewirken.

Kein Wunder, dass der Erfolg ausser für einfache Regelaufgaben unter starker externer Dämpfung eher mässig ist, und alternative Ansätze gesucht werden.

Beim *Assoziativspeicher* dagegen findet sich in der Schwellwertreduktion ein **dämpfendes inneres** Merkmal. Im Gegenzug können aber sogar „unstetige“ Sachverhalte **direkt eingelernt** werden, und auch erfolgreich reproduziert, weil durch die innere Dämpfung auch eine innere Selektionsgenauigkeit bedingt ist: Schon ein einzelnes Bit kann problemlos einen beliebigen Datensatz von irgendeinem anderen unterscheiden.

Fassen wir zusammen: traditionelle KNN werden gezwungen, Stücke von Geschichten möglichst wortgetreu wiederzugeben, können dies aber im allgemeinen eigentlich nicht gut.

Besonders bei unstetigen Funktionen wie Texten haben Assoziativmatrizen hier offenbar eine bessere Prognose.

Warum? Eben weil man Ihnen dank der expliziten Lernregeln auch explizit Wissen beibringen kann.

...wie neuronale Netze...

Wissen beibringen – und dann auch wieder Wissen abzufragen – das bedeutet normalerweise, jemandem etwas zu erklären und dann Fragen zu stellen, deren Erläuterung durch die befragte Person einen Rückschluss auf das Wissen oder dessen Nutzung erlaubt.

Dabei ist eine solche Frage oft *eher kürzer* als die Antwort – *nicht* wie bei einem KNN zur Aktienprognose oder Klassifizierung *umgekehrt*.

Einer Nachbildung dieser natürlich erscheinenden Eigenart – dass eine „korrekte“ Antwort im Mittel gerade dieselbe „Informationsmenge“ wie die Frage erwartungsgemäss umfassen kann oder soll – allein schon weil so automatisch

eine Möglichkeit gegeben ist, den Antwortumfang *ohne weiteren Formalismus* implizit als festgelegt zu betrachten – kann sinnvoll sein.

Antwortumfang, Symmetrie

Man kann hier bemerken: Dieser Gleichrangigkeit von Fragen und Antworten findet ihren Niederschlag auch in der diesbezüglichen Symmetrie der Lernregel (L) und es ergibt sich automatisch die Idee, eine Anwort selbst wieder als Frage zu begreifen und so zu einer Interaktion mit dem Matrixspeicher zu gelangen.

(Später könnte man dann allerdings auch noch versuchen, auch die Interpretation von Zahlenfolgen – also wieder die Prognose von Aktienkursen – in Form einer textuellen Beschreibung dem Assoziativspeicher wieder zuzuführen, wenn dies gewünscht ist.)

Bisher wurden assoziative Strukturen allerdings zumeist als spezielle künstliche neuronale Netze angesehen, dementsprechend ergeben sich als typische Nutzungsarten auch die klassischen Anwendungsgebiete dieser Netze:

Mustererkennung durch Merkmalsextraktion, Musterergänzung und darauf aufbauend Klassifikationsaufgaben.

Die in das Netz eingegebenen Strukturen sind hierbei zunächst grundsätzlich voneinander unabhängig, das heisst ohne eine sequentielle Ordnung.

Die eben beschriebene Interaktivität ist daher als ein Novum anzusehen.

...zählen können.

Einen weiteren Schritt in der Nutzung assoziativer Systeme für „traditionelle“ Speicherungsanwendungen ging also Dierks, wenn er in *VidAs 3* [(3)] eine sogenannte *Abfolgematrix* nutzt, um in geordneter Weise also eine **Sequenz** von Befehlen aus einer zweiten Matrix abrufbar zu machen, was die Basis seines programmierbaren Vidas-Systems bildet.

Die dort verwendeten Sequenzbildungen sind allerdings von einfacher Art und nicht mit den intrinsisch bedeutungstragenden Merkmalen der Nutzdaten verbunden, sondern werden

- entweder im Gegenteil von diesen getrennt [(4)] , oder
- *sind* zwar tatsächlich die Nutzdaten im Sinne einer Zählfolge entsprechend dem Konzept der Ordinalzahlen (so schon in Abfolgematrix und Fortsetzungskern, jedoch auch in den durch die Vidas-Befehle *&zfolge* und *&zkreis* erzeugten offenen und geschlossenen Folgen) – als solche allerdings nicht frei wählbar.

(3) [Dierks 05] S. 55-57

(4) Abfolgematrix A vs. Befehlsmatrix B (vgl. [Dierks 05] S. 57, dort Abb. 41) beziehungsweise klar lokalisierter *Fortsetzungsfaden* (so in [Bentz, Dierks 12] Kap. 5.16 „Maskieren und Vereinen“)

Rekapitulation

Es bleibt festzuhalten: Assoziativspeicher sind Speicher, welche also solche zunächst nicht binäre und lückenlos geordnete Adressen verarbeiten und Inhalte unter diesen Adressen vermerken beziehungsweise von diesen zurückliefern, sondern direkt über Inhalte adressiert werden können und andere Inhalte zurückliefern.

Eine derartige Zuordnung $F \rightarrow A$ („Frage zu Antwort“) bezeichnen wir als Assoziation (F, A) und reden davon, dass ein Assoziativspeicher „Assoziationen lernt“ und später „Fragen beantwortet“.

Die geeignete Kodierung der Fragen und Antworten ist eines der Untersuchungsgebiete der sogenannten *Assoziativen Programmierung* [(5)] ebenso die Aufgabe, Programm-Sequenzen in solchen Speichern abzulegen und durch eine Assoziativmaschine ausführen zu lassen.

Hier, bei Sequenzen von Befehlen oder dann zum Beispiel *Sätzen*, knüpft auch die Textspeicherung an.

Während die *Assoziationen* zunächst ungeordnet sind, wird in Sequenzen oder auf anderem Wege, konkret durch Zählfolgen, eine Ordnung möglich.

Eine erste Möglichkeit zur Speicherung von Sequenzen (Programmen oder Texten) ergibt sich somit natürlicherweise:

Wenn man Adressen in geeigneter Form als „Fragen“ verstehen und *codieren* kann, so lassen sich die Assoziativspeicher auch direkt als Datenspeicher nutzen.

Eigenschaften solcher Datenspeicherung und Varianten von derartigen „Adressfragen“ sowie *Alternativen zu diesem Konzept* bilden die Basis dieser Arbeit.

Im Ergebnis können Daten aus ihren Indizierungen rekonstruiert werden, aber auch Erkenntnisse über deren Struktur gewonnen werden, welche in statistischem Zusammenhang mit semantischen Begriffen stehen [(6)] .

(5) [1] (Website zur Assoziativmaschine), [Bentz, Dierks 12]

(6) vgl. [Ackermann 00]; eine Realisierung ist bereits die Context-Funktion des Programmes [5]

3.2. Formalisierung der Matrixoperationen und der Textspeicherung

Wir wollen zunächst die Struktur des Assoziativspeichers und dann eines Textes formalisieren.

Dazu geben wir einige Definitionen und Regeln formelhaft an, so dass wir die Operationen mit Assoziativspeichern unzweideutig bezeichnen können.

Diese sind als binäre Matrizen zunächst mathematische Objekte, welche in unseren praktischen Versuchen Realisierung bzw. Simulation in Computerprogrammen erfahren [(7)] .

Klar zu unterscheiden davon sind die Objekte O der Realität, welche eine Menge von Merkmalen zugeordnet bekommen soll. Ob ein bestimmtes Objekt dabei zum Beispiel *rund* oder *klein* ist, oder aber *an einer bestimmten Stelle den Text* <**AB**> enthält, ist

- in den ersten Fällen vordergründig nicht durch uns betrachtet [(8)]
- im letzten Falle klar definiert und auch die Basis unserer Textspeicherung.

Verschiedene Klassen von Merkmalen werden erst später genauer definiert.

3.2.1. Assoziativspeicher

Definitionen, Objekte, Merkmalsvektoren

Menge der Merkmale eines Objektes:

$$codes[O] = \{\ldots\} \subset \mathbb{N}_0$$

Merkmal mit Index i:

$$i \in \mathbb{N}_0,$$

Spaltenvektor F („Frage“) über die Zeilen i:

$$F = (F_i) = \left(1_{[i \in codes[F]]}\right)$$

und analog Zeilenvektor A („Antwort“) über die Spalten j:

$$A = (A_j) = \ldots$$

(7) Dabei werden teilweise Speicherstrukturen verwendet, welche nicht einer direkten Matrixdarstellung entsprechen; z.B. entlang der „Feld-Zeiger Darstellung der dynamischen Matrix“ aus [Heitland 94] S. 67

(8) weitere Beispiele finden sich in bekannten Merkmalen, wie sie bei der Bildverarbeitung aus Pixeldaten gewonnen werden oder auch in abstrakter Form (Sammlung von Geschmacks-*Eindrücken* bei Schokolade) bei Dierks (Seminarunterlagen bzw. [Bentz, Dierks 12])

Assoziation τ bestehend aus einer zu lernenden *Frage* $\rightarrow$ *Antwort*:

$$\tau_t = (F^t, A^t) \tag{t}$$

Memory a zum Zeitpunkt t [(9)] :

$$(a_{ij}^t)$$

Lernen

Kumulierende Lernregel:

$$(a_{ij}^{t+1}) = (a_{ij}^t \vee F_i^t A_j^t) \tag{L}$$

Lernregel mit Verdrängung (Vergessen):

$$(a_{ij}^{t+1}) = (\overline{F}_i^t a_{ij}^t \vee F_i^t A_j^t) \tag{K}$$

(vgl. Dierks) [(10)]

Zur Vereinfachung (und entsprechend die vorigen Formeln) auch notiert als:

$$(a_{ij})' = (\overline{F}_i a_{ij} \vee F_i A_j) \tag{K'}$$

Auslesen

Schwellvektor:

$$S[F] = (S_j[F]) = (\sum_i F_i a_{ij}), \tag{S}$$

entsprechend einer Multiplikation des transponierten F^T an a zum Zeitpunkt t, daher auch:

$$s^t(F) = F^T a^t \tag{s}$$

Reduktion auf Maximalschwelle liefert $A[F]$:

$$A[F] = (A_j[F]) = (1_{[S_j[F]=\max_j S_j[F]]}), \tag{R}$$

auch notiert als:

$$r^t(F) \tag{r}$$

---

(9) mit $(a^0) = (0)$

(10) [Dierks 05] S. 16, S. 56 u. 61; dort heissen die Regeln *Langzeit-Lernregel* und *Kurzzeit-Lernregel*

Bemerkungen

- Für L-Speicher gilt offensichtlich [11] :
  1. $$\left|A[F^t]\right| \geq \left|A^t\right|, \tag{3.1}$$

     denn es können nicht mehr Merkmale aktiviert werden, als in der Frage vorliegen. Aber genau diese Merkmale sind auch im Memory durch L für jedes $i \in codes[F]$ in den jeweiligen Spalten j belegt, so dass dort die maximal mögliche Summe s auch tatsächlich erreicht wird. Infolge wird genau auf dieser Schwelle reduziert. Ensprechend ergibt sich auch zumindest an diesen Stellen j der Maximalwert s, und j wird somit in (A_j) markiert.

     Es können zusätzliche Positionen k („Lese-Fehler“) resultierend aus überlappenden *codes* markiert sein.
  2. Die Lern-Operation ist idempotent und die Reihenfolge der Einspeicherung ist nicht relevant.
- Die Kurzzeit-Lernregel K bewirkt das vollständige Eintragen des A^t in a^{t+1} genau in den Zeilen i, in welchen F_i^t gegeben ist; ansonsten wird a^t beibehalten. Damit wird immer

  $$r^t(F^t) = A^t \tag{3.2}$$

  korrekt geliefert; ein korrektes Ergebnis $A^u = A[F^u]$ für $u < t$ ist jedoch nicht mehr gewährleistet, falls $codes[F^u] \cap codes[F^t] \neq \varnothing$.
- s ist linear,
- r ist nicht-linear.

3.2.2. Texte

Konstante Blockgrösse

Unter Texten verstehen wir im einfachsten Falle eine Sequenz V^i von T Stück, also $0 \leq i < T$, Vektoren [12] gleicher Länge c über einem Alphabet C, welches b unterscheidbare Zeichen enthält.

Wenn das Fortschreiten von einer Textzeile i zur folgenden Zeile $i+1$ durch die Assoziation τ_i beschrieben wird, so haben wir:

$|\{\tau_t\}| \leq T-1$ Stück Assoziationen mit $0 \leq t < T-1$.

[11] $|V|$ bezeichnet hier die Anzahl gesetzter Bits im Merkmalsvektor V, nicht seine Länge. Dies geht allerdings konform mit der Interpretation der Menge der Merkmale eines Objektes als *Indexmenge*

[12] welche Zeilen oder Worte *genannt werden*, obwohl Trennung an anderen als Wort- oder Zeilen-Grenzen vorgenommen werden; siehe das Beispiel in Kap. 3.5.2 auf S. 58

(Selbstverständlich lässt sich jeder „irgendwie wohldefinierte“ Text in dieser Form darstellen, gegebenenfalls durch Auffüllen mit einem zusätzlichen Füllzeichen)

3.2.3. Textspeicherung

Die Aufgabe der Textspeicherung in Assoziativspeichern, welche ja nur mit den Assoziationen $\{\tau'_{t'}\}$ „befüllt“ werden, ist im Einzelfall immer dann gelöst, wenn zu einem gegebenen befüllten Speicher $a := (a^{T'-1})$, also nach dem Einspeichern aller $T'-1$ Assoziationen zu einem Text,

1. jedes $F^i[i \in \{t'\}]$ (Satzschlüssel) angegeben werden kann als

$$F^i = f(a, \{F^j\}, \{A^j\}) \,\text{mit}\, j < i \tag{f}$$

   - $B_i := (a, \{F^j\}, \{A^j\})[j < i]$ ist dabei die im Laufe des Auslesens von Anfang an inzwischen bereits gesammelten Informationen.
   - Wenn wir nicht alle Informationen vollständig zugreifbar aufbewahren wollen oder wenn wir einmal mitten im Text mit dem Auslesen beginnen, kann es sinnvoll sein, einen Rückgriff auf einige wenige Elemente zu beschränken.

$$B_{i,k} := (a, \{F^j\}, \{A^j\}) \,\text{mit}\, i-k \leq j < i \tag{B}$$

     soll daher das nur k Schritte kurze *Backlog* sein (also ist $B_{i,i} = B_i$).

2. und die Abbildung (Dekodierung)

$$d : \begin{cases} \{\tau'_{t'}\} \to (V^{t'}) \\ \{(F^{t'}, A^{t'})\} \mapsto (V^0, \ldots, V^{T'-1}) \end{cases} \tag{d}$$

   angegeben ist, sowie

3. (Speicher-Validität)

$$r^{t'}(F^{t'}) = A^{t'} \tag{v}$$

   gilt.

Korrekturfunktionen beim Lesen

- Möglicherweise erhalten wir den korrekten nächsten Schlüsselvektor erst nach einer Korrektur durch eine Validierungsabbildung v (abgeschwächte Speicher-Validität):

$$v(r^{t'}(F^{t'}), B_{\ldots}) = A^{t'} \tag{w}$$

Speicher mit Startschlüssel

Für τ_0 ist $F^0 = f(a, \{\}, \{\})$ nur von a und f selbst abhängig. Wenn F^0 *nicht* von a abhängig ist, ist es also eine Konstante von f. Ansonsten muss ein Verfahren angegeben sein, wie es aus a zu bestimmen ist.

Verschiebungen

Verschiebung, Transformation oder auch *Translation* heisst eine Bijektion

$$l : \{\tau_t\} \rightarrow \{\tau'_{t'}\} \ , \tag{l}$$

welche zu einem gegebenen und parametrisierten Verfahren die Einspeicherung der $\{\tau'_{t'}\}$ schliesslich ermöglicht, wo dies für die $\{\tau_t\}$ unter Umständen nicht gelang. Wenn zu $\{\tau_t\}$ eine Dekodierung d existiert, dann existiert trivialerweise zu $\{\tau'_{t'}\}$ eine Dekodierung d'.

3.3. Allgemeines zur Textspeicherung

Wie aus dem Beispiel im Abschnitt 3.1 ersichtlich, sind eindeutige „Schlüssel“ (das sind die Fragen F^t) eine Voraussetzung für ein erfolgreiches Durchlaufen einer Assoziationskette.

Wo diese Eindeutigkeit nicht gegeben ist, können wir sie eventuell herstellen;

Zunächst greifen wir in Kap. 3.4 in Erweiterung des Ansatzes der *Abfolgematrix* aus [Dierks 05] auf – allerdings im Gegensatz dazu auch überlappende – sogenannte Zähl- oder Index-Folgen zurück, welche zunächst nach einer vorgegebenen Bitdichte „zufällig“ gewählt werden, das heisst, wir verwenden Folgen mit einer festen Anzahl von vorhandenen Einzelmerkmalen.

Weiterhin bietet sich aufgrund unserer Untersuchung von „langen Sequenzen“ [(13)] auch die Prüfung solcher Folgen als Indexfolgen an.

Als nächste Alternative bietet sich dann folgerichtig an, die Fragevektoren F^t der Assoziationen aus dem Text selbst zu beziehen, also diesen Text in geeigneten Einheiten als Sequenz aufzufassen – was jeder Text ja auch natürlicherweise ist.

Es kann jeweils auf Zeichen- oder Wortebene sowie dann immer durch Positionskodierung oder überlappende Tupel-Kodierungen vorgegangen werden, wovon wir eine Variante betrachten.

[(13)] vgl. Kap. 4.3

3.3.1. Grundüberlegungen

Nach der aus der Materie *Zeichenfolge* und *Matrixspeicher* eher automatisch folgenden Definition der Aufgabe *Textspeicherung in Assoziativspeichern* wie im Abschnitt 3.2 angegeben, ergaben sich die nachfolgenden Fragen zwar ebenfalls beinahe von selbst, jedoch nicht sofort deren Antworten.

Im Rahmen dieser Arbeit wurde daher einzelnen dieser Fragen nachgegangen, wobei

- ⋆ einige Items die Grundlage der nachfolgenden Abschnitte bilden,
- \- während andere maximal eine übersichtsartige Behandlung im Sinne eines Ausblicks im Kapitel 4 erfahren können.

Mit dem Mittel des Assoziativspeichers ausgerüstet, lassen sich zur Textspeicherung ohne besondere Reihenfolge oder Anspruch auf Vollständigkeit einige Fragen formulieren:

- ⋆ Wie lässt sich ein Text T durch eine Menge $\{\tau_t\} = \{(F^t, A^t)\}$ eineindeutig beschreiben?
- ⋆ Ist $|\{\tau_t\}|$ eher gross oder klein?
- ⋆ Verwendet man *einen* oder *mehrere* Assoziativmatrizen zur Speicherung?
- ⋆ Ist L oder K oder eine Kombination beider zu verwenden?

Je nach Wahl der Verfahren können sich dann weitere Fragen ergeben, welche je nach Anwendungszweck mehr oder weniger relevant sein können:

- ⋆ Soll man über alle τ_t: $|F^t| \approx const$ gewährleisten? Ebenso $|A^t| \approx const$?
- ⋆ Was wird zur Rekonstruktion des Textes aus (a) benötigt werden? Im folgenden werden hier Möglichkeiten aufgezeigt, welche im wesentlichen auf eine der folgenden Varianten hinauslaufen:
  - ⋆ Es wird eine *„Zählweise“* $count : [n \in N] \rightarrow [c(n) = F^n]$ verwendet; das bedeutet, man benötigt eine kanonische Darstellung von Zahlen $t \in N \rightarrow F^t$, zumindest jedoch eines t_0 als F^0.
  - \- Man benötigt die *erste Zeile* (oder einen *Namen*) des Textes, vergleichbar einem Schlüssel, als F_0.

  Wir wollen dieses t_0 oder F_0, wenn wir es als gleichwertig betrachten möchten, auch q_0 („die erste Frage“) nennen.

  Im einfachsten Falle ist hierzu dem Text eine Konstante voranzustellen, welche allerdings nicht einer Transformation nach (l) unterliegen darf, also muss gelten $q_0 = l(q_0)$.
- \- In den Varianten, welche ein $count[n \in N]$ *nicht* benötigen: Wie erhält man das $q^{t+1}(t)$? Oder gar eine Menge $\{q_i^{t+1}\}^t$?

- Findet man die $\tau_i \in \{\tau_t\}$ in einer bestimmten Reihenfolge (entsprechend einer noch zu klärenden Lokalitätsbeziehung in T) oder „nur insgesamt“?
- Eng verwandt damit ist auch die Frage: Kann man (wie beim Aufschlagen eines Buches) auch irgendwo „in der Mitte“ (weiter-)lesen?
- Dies steht natürlich auch im direkten Zusammenhang mit der Durchsuchbarkeit des Textspeichers.
- Beziehungsweise: Findet man *überhaupt* die τ_t oder „nur“ die Menge aller $\{A^t\}$ oder $\{F^t\}$, welche dann zur Rekonstruktion von T ausreichen?
- Wie erkennt man, ob man alle τ_t gefunden hat?
- Sowie schliesslich (was hier nicht betrachtet werden soll): In welchem Ausmass bleibt die Eigenschaft der *Störunanfälligkeit* der Assoziativspeicher bestehen? Lassen sich bei Störungen noch Teile des T rekonstruieren?

An einige dieser Fragestellungen wird sich jetzt herangetastet.

Bemerkung Zur Darstellung von im praktischen Experiment durchgeführten Untersuchungen wollen wir einige Bemerkungen machen:

Die später verwendeten Prinzipien sollen anfangs möglichst anschaulich dargestellt werden können; andererseits sollen zumindest die Elementareigenschaften der Assoziativspeicher sichtbar werden.

Wir folgen der Forderung nach einer spärlichen Kodierung der Daten bei assoziativer Speicherung [(14)] , welche unter anderem das Auslesen der Matrix *besonders schnell* ermöglicht, [(15)] , hier jedoch erst einmal nicht vollständig:

Nach näherungsweisen Berechnungen genügt ein Merkmalsvektor einer Spärlichkeitsanforderung frühestens etwa bei einer Bitdichte geringer als 1% bis 1‰ [(16)] .

Da wir andererseits *mindestens* 2 gesetzte Bits benötigen werden, um Überlagerungseffekte miteinzubeziehen, ohne dass zwei verschieden genutzte Vektoren identisch würden, benötigt man entsprechend mindestens eine Vektorlänge von 200 bis 2000 Bits.

Tatsächlich angemessen wären wohl eher 5000 – man vergleiche auch die im VidAs System[Dierks 05] derzeit implementierte maximale Matrixgrösse $n = 2000$, wo wir bei Experimenten zur Mustererkennung oft im Bereich $n = 1000$ bis 2000 gearbeitet haben – allerdings ist eine Darstellung im Detail derartig dimensionierter Matrizen auf dem Papier nicht einfach möglich.

---

(14) vgl. [Bentz et al. 89], auch wegen der zu erreichenden nutzbaren Speicherkapazität

(15) [Bentz et al. 89]; auch [Palm 13]: „If the input patterns are sparse it is even faster, since the number of operations is simply proportional to the number of 1-entries in an input pattern.“

(16) [Hagström 96] S. 40 setzt die Belegungsdichte im Bereich weit unter $1E-4$ bis hin zu $1E-11$ an

Später werden wir die einfachen Untersuchungen erneut auf derart langen (und längeren) Bitvektoren (und auch mit mehr gesetzten Bits) durchführen,

- um nicht nur Spezialfälle zu betrachten,
- auch um überhaupt mehr Nutzinformationen einspeichern zu können.

n,k: Dimensionierung der einfachen Beispiele

Fürs Erste jedoch zur Veranschaulichung beschränken wir uns – soweit nichts anderes ersichtlich – auf Matrizen der Dimension $n \times n = 10 \times 10$ mit $k = 2$ gesetzten Bits in den F^t, A^t. Zur einfachen Darstellung wollen wir in den Beispieldarstellungen auch davon ausgehen, dass n ein Vielfaches von k ist, was insbesondere dann angemessen ist, wenn die sogenannte „Buchstabe-auf-Platz-Kodierung“ Verwendung findet, da dann jeder Zeichenposition innerhalb einer als von konstanter Länge gedachten Zeichenfolge ein immer gleichlanger Teilbereich des gesamten Merkmalsvektors zugeordnet ist.

3.3.2. Ein erster Ansatz

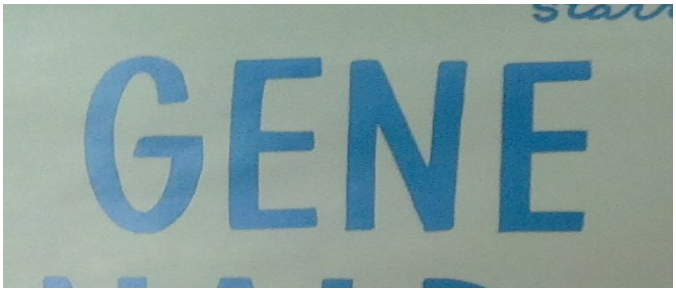


Abbildung 3.1. *Ein Text[6]*

Im Bild 3.1 zu Erkennen ist der Text <**GENE**> (ein Vorname), welcher in eine Matrix gespeichert werden soll. Wir verwenden angelehnt an die ASCII-Kodierung zunächst die einfache Binärcodierung. Zur Darstellung des Prinzips wird in diesen einfachen Beispielen auch immer auf eine Unterscheidung von Gross- und Kleinbuchstaben verzichtet.

$A \mapsto$ %00001
$\vdots$
$E \mapsto$ %00101
$\vdots$
$G \mapsto$ %00111
$\vdots$
$N \mapsto$ %01110
$\vdots$
$Z \mapsto$ %11010

Der Assoziativspeicher wird entsprechend der Grösse der Binär-Vektoren als $5x5$-Matrix angelegt.

Darstellung im Programm „igehirn“ Das Programm hat seinen Namen aus der Kombination „interaktiv“ und „Gehirn“ – es bietet *Interaktion* durch klickbare Merkmale auf Vektoren und Sequenzen von Vektoren, welche automatisch geprüft werden. Die korrekte oder nicht korrekte Speicherung wird farblich visualisiert. Dabei wird ein Unterprogramm verwendet, welches z.B. auch in kleinen Robot-Fahrzeugen Steueraufgaben erfüllen kann, weshalb das entsprechende Modul anfangs den Namen *gehirn* erhielt.

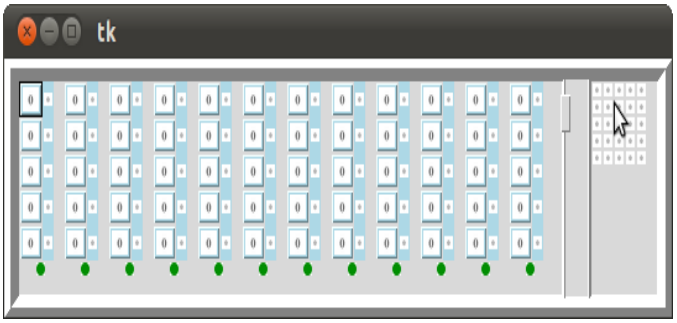


Abbildung 3.2. *igehirn.py mit leerer* 5×5*-Matrix*

In Abbildung 3.2 ist das Programm mit einer $n^2 = 5 \times 5$-Matrix gestartet. Von links nach rechts kann eine Sequenz eingetragen werden, welche dann in *überlappend versetzter Paarung* (vgl. auch Abbildung 3.8) in den Speicher eingeschrieben („gelernt“) wird [17] .

Assoziationen in eine Reihenfolge bringen Da nicht unabhängige Assoziationen, sondern ein geordneter Text wieder abgerufen werden können soll, muss irgendwie die Ordnung berücksichtigt werden. Es bieten sich zwei Möglichkeiten einer Ordnung an:

1. Explizite Ordnung im Sinne einer Abbildung $t \mapsto F^t$, sowie
2. die Nutzung der impliziten Ordnung $F^t \mapsto F^{t+1}$. [18]

[17] zur genaueren Beschreibung der Bildschirmdarstellung siehe den Ausschnitt der Programmdokumentation im Anhang B.2

[18] Im ersten Falle muss die Zählfolge entweder deterministisch berechenbar sein oder selbst mit gespeichert werden; im zweiten Fall ist zu gewährleisten, dass die implizite Ordnung eine eindeutige Abfolge der F^t zu bestimmen in der Lage ist. Dazu muss hinreichend viel Information gegeben sein, welche – falls “aufgeprägt“ auch nicht durch die Nutzdaten maskiert wird. Weiterhin ist noch ein Startwert notwendig, da ja die implizite Ordnung $F^t \mapsto F^{t+1}$ erst *innerhalb*, also *zwischen* den F^t, F^{t+1} wirkt, aber nicht davor.

Wir beginnen mit dem einfachen Beispiel:

Eingespeichert werden soll nun in die $5x5$-Matrix, soweit dies jeweils möglich ist: <**G**> <**GE**> <**GEN**> <**GENE**>

Wir verwenden zunächst die oben beschriebene Kodierung.

Es ergeben sich im Fall 1 die zu speichernden Vektoren

F^t	A^t		F^t	A^t
<**A**>	<**G**>		%00001	%00111
<**B**>	<**E**>	bzw.	%00010	%00101
<**C**>	<**N**>		%00011	%01110
<**D**>	<**E**>		%00100	%00101

.

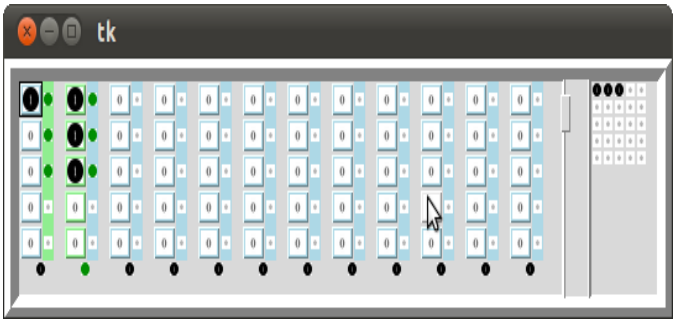


Abbildung 3.3. *igehirn.py mit* $\{\tau_t\} = \{(A, G)\}$

In Abbildung 3.3 ist dargestellt, wie die erste Assoziation in das Programm eingetragen wird (grosse Punkte). Rechts ist die Matrixbelegung sichtbar. Mit dem Vektor <**A**> „von links“ anzufragen liefert auch <**G**> , wie aus den drei kleinen Punkten neben den drei grossen Punkten ersichtlich ist.

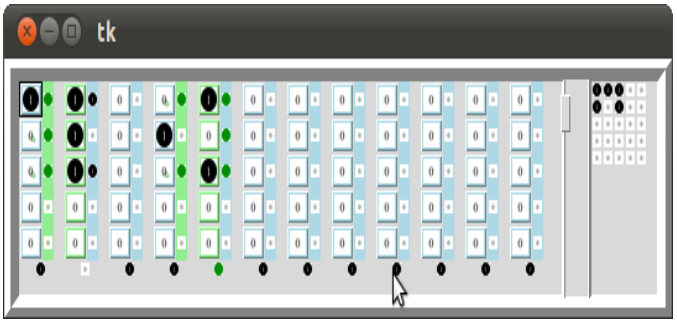


Abbildung 3.4. *igehirn.py mit* $\{\tau_t\} = \{(A, G), (B, E)\}$

Abbildung 3.4 zeigt auch, dass das zweite Paar noch eingespeichert werden kann.

Der Versuch in Abbildung 3.5 zeigt dann jedoch schon, dass es so nicht weiter funktionieren wird: Hier sind die Assoziationen noch nicht gelernt (nur die

F^2, F^3 sind eingetragen, nicht die zugehörigen A^t), dennoch zeigt die Abfrage bereits (und zwar mit Schwelle 2) das oberste beziehungsweise rechteste Bit $R_0^2 = 1$ fest gesetzt, welches im zu <**N**> gehörigen A^2 jedoch 0 ist.

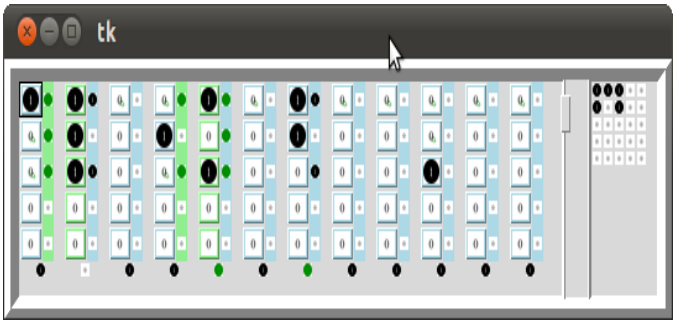


Abbildung 3.5. *igehirn.py mit* $\{\tau_t\} = \{(A, G), (B, E)\}$ *und Abfragen von C*

Als Ursache ist anzugeben, dass einerseits die Belegung des A^0 im fraglichen Bereich natürlich zu dicht ist, andererseits die Selektion durch F^0 beziehungsweise F^1 mit jeweils $|F^t| = 1$ „zu einfach“ ist. Besonders sichtbar wird dies schliesslich, wenn man tatsächlich alle Vektoren einschreibt und prüft, wie in Abbildung 3.6 vorgenommen. Die zusätzlich zurückgelieferten 1-Bits verfälschen *jede* Assoziation. Tatsächlich wird auf alle Fragen nur noch <**O**> zurückgeliefert (überall Punkte in den Kontrollspalten).

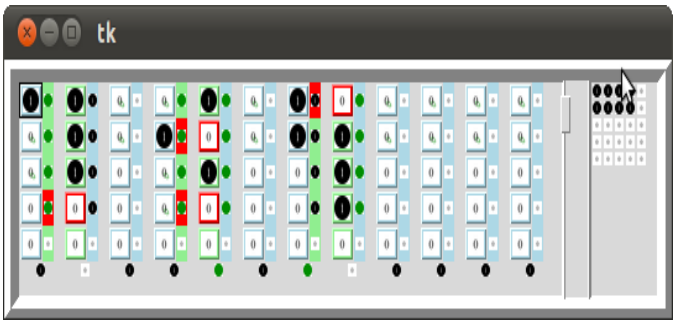


Abbildung 3.6. *igehirn.py mit* $\{\tau_t\} = \{(A, G), (B, E), (C, N)\}$

Spezialfall k=1 Es ist dagegen natürlich trivial ersichtlich, dass mit einer Kodierung

$A \mapsto \%00001$, $B \mapsto \%00010$, $C \mapsto \%00100$, $D \mapsto \%01000$, $E \mapsto \%10000$

(für die F^t) die geforderte Speicherung zu leisten wäre. Allerdings ist auch offensichtlich, dass bei einer $n \times n$-Matrix *immer gerade maximal* n *Werte* gespei-

chert werden können, und dies – solange möglich – auch exakt einer Speicherung in einem unär adressierten [19] jedenfalls aber konventionellen RAM-Speicher entspräche. *Wir gehen daher hier nicht weiter darauf ein.*

Kombinatorische Betrachtungen zu Spärlichkeit

Wie soeben dargelegt, rühren die Probleme der Speicherung offenbar daher, dass die Belegungsdichte der F^t und A^t

1. zu hoch und
2. nicht konstant

ist. Im Beispiel war die Belegungsdichte [20] im Mittel $m_p \approx 0.46$ mit einer Streuung von $s_p \approx 0.16$.

Ein konstantes p lässt sich bekanntlich leicht erreichen, indem die $\binom{n}{k}$ Auswahlen genutzt werden.

n	k	$\binom{n}{k}$	p
1	n.a.	n.a.	n.a.
⋮	⋮	⋮	⋮
7	3	35	0.429
8	2	28	0.250
9	2	36	0.222
10	2	45	0.200
⋮	⋮	⋮	⋮
26	1	26	0.038
27	1	27	0.037
28	1	28	0.036
⋮	⋮	⋮	⋮

Tabelle 3.1. *minimale Matrixgrössen zur Überdeckung von 26 Zeichen*

Die Tabelle 3.1 zeigt zu einer gegebenen Matrixgrösse n, soweit möglich die kleinste Zahl k (wieviele Bits gesetzt sein müssen), um mit den resultierenden kombinatorischen Möglichkeiten ein Alphabet von 26 Zeichen überdecken zu können. p ist die daraus resultierende Bitdichte in den Vektoren.

Umgekehrt ist zu einem vielleicht gewünschten $k = 2$ auch das minimale n abzulesen, so dass eine direkte Kodierung möglich ist. Wenn man den zweiten auftretenden Eintrag wählt, bei dem $\binom{n}{k} \geq 26$ ist, also $(n = 8, k = 2)$ und nicht etwa $(n = 25, k = 2)$, so ist immer noch $p \approx 0.25$.

[19] wie durch eine Selektionsleitung nach einem Multiplexer

[20] einmal vorausgesetzt, dass jeder Buchstabe mit gleicher Wahrscheinlichkeit verwendet wird

Zur Veranschaulichung zeigen wir einmal alle Vektoren von $(n = 12, k = 3)$ mit ebendiesem Wert von $p = 0.25$ in Abbildung 3.7.

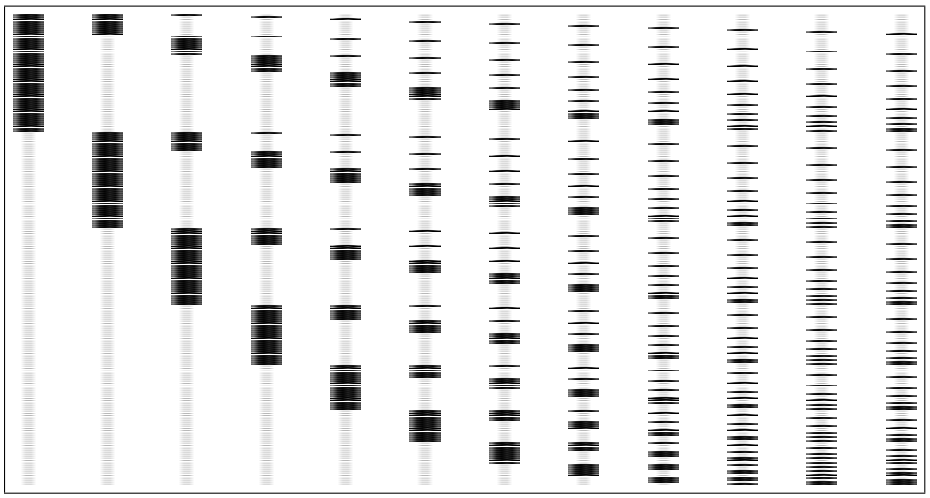


Abbildung 3.7. *Alle* $\binom{12}{3}$ *Möglichkeiten, geordnet*

Da im Ausgabevektor ja auch mehrere Bits gesetzt sind und zu mehrfachen Einträgen führen – man sich also das Bild etwa mehrmals leicht verschoben oder gedreht transparent überlagert vorstellen muss – sind offenbar geringere Bitdichten anzustreben. Bei $(n = 25, k = 2)$ mit $p = 0.08$ wird allerdings nur etwa jeder zehnte Vektor benötigt, und man kann nicht sinnvoll direkt mit der natürlichen Ordnung der zwei-elementigen Auswahlen arbeiten.

Zum Vergleich bestimmen wir analog Kodierungsmöglichkeiten für Zwei-Buchstaben-Paare, welche wir später verwenden werden (Tabelle 3.2), sowie für Tripel (Tabelle 3.3); hiernach ergibt sich für Paare bei $(n = 38, k = 2)$ eine Belegung der Vektoren von $\approx 5\%$, was noch hoch erscheint.

Erst für mindestens 3-Tupel mit $k = 2$ Kodierpositionen wird die gewünschte Grenze der Spärlichkeit von etwa 1% erreicht.

Tatsächlich sind die Werte in der Realität oft günstiger, wenn man z.B. Gross- und Kleinschreibung sowie Satzzeichen und Zahlen berücksichtigt:
Während dies zwar die Erkennung von Schreibvarianten – zum Beispiel bei einer fehlertoleranten *Suche nach ähnlichen Dateinamen* – erschwert, ist doch die zusätzliche Information, welche mit Satz- sowie Wort-Anfängen und -Enden korreliert ist, sowohl für die *Textspeicherung* als auch für darauf aufsetzende *adaptive Steuerungsstufen* natürlicherweise notwendig beziehungsweise von Vorteil.

n	k	$\binom{n}{k}$	p
1	n.a.	n.a.	n.a.
⋮	⋮	⋮	⋮
12	5	792	0.417
13	4	715	0.308
14	4	1001	0.286
15	4	1365	0.267
⋮	⋮	⋮	⋮
17	3	680	0.176
18	3	816	0.167
19	3	969	0.158
⋮	⋮	⋮	⋮
38	2	703	0.053
39	2	741	0.051
40	2	780	0.050
⋮	⋮	⋮	⋮
676	1	676	0.001
677	1	677	0.001
678	1	678	0.001
⋮	⋮	⋮	⋮

Tabelle 3.2. *minimale Matrixgrössen zur Überdeckung von 26^2 Zeichen*

n	k	$\binom{n}{k}$	p
1	n.a.	n.a.	n.a.
⋮	⋮	⋮	⋮
17	7	19448	0.412
18	6	18564	0.333
19	6	27132	0.316
20	6	38760	0.300
21	5	20349	0.238
22	5	26334	0.227
23	5	33649	0.217
⋮	⋮	⋮	⋮
28	4	20475	0.143
29	4	23751	0.138
30	4	27405	0.133
⋮	⋮	⋮	⋮
49	3	18424	0.061
50	3	19600	0.060
51	3	20825	0.059
⋮	⋮	⋮	⋮
188	2	17578	0.011
189	2	17766	0.011
190	2	17955	0.011
⋮	⋮	⋮	⋮

Tabelle 3.3. *minimale Matrixgrössen zur Überdeckung von 26^3 Zeichen*

Im realen Text aus Abschnitt 3.5.2 ergibt sich zum Beispiel ein Alphabet zu 89 verschiedenen Zeichen; es gelten Tabelle 3.4 (Paare); Tabelle 3.5 (Tripel). Mit $p(n = 127, k = 2) = 0.016$ wird die Prozentgrenze bei Paarkodierung bereits in etwa getroffen, während bei $3 - Tupeln$ schon die Wahlmöglichkeit zwischen $(n = 163, k = 3)$-Codes mit ähnlicher Dichte und $(n = 1188, k = 2)$-Codes mit einer Dichte $p \approx 1‰$ besteht. Die durch verschiedene Kombinationen von Tupellänge l_t mit Bit-Anzahlen k erreichbaren Teil-Vektorlängen l_w:

$$\{(l_t = 2, k = 2) \mapsto l_w = 127, (l_t = 3, k = 2) \mapsto l_w = 1188, (l_t = 3, k = 3) \mapsto l_w = 163\}$$

ermöglichen bereits Variationen der Kodierung in einem weiten Rahmen. Wenn dann Worte oder Zeilen z.B. der Urtext-Länge $c = 10$ als Sequenzen *über dem Tupelalphabet mit jetzt Länge* $c_t = \lceil 10/l_t \rceil$ betrachtet werden, wobei jedes l_t-Tupel auf seiner Position markiert wird, lässt sich die Gesamt-Vektorlänge l_v und damit n zwischen $l_v = 100 \cdot c_t \approx 100 \cdot \frac{c}{l_t} \approx 300$ und $l_v = 1000 \cdot c_t \approx 3000$ wählen, um eine dem Gesamttext angemessene Matrixgrösse zu finden.

n	k	$\binom{n}{k}$	p
1	n.a.	n.a.	n.a.
⋮	⋮	⋮	⋮
16	6	8008	0.375
17	6	12376	0.353
18	5	8568	0.278
19	5	11628	0.263
20	5	15504	0.250
⋮	⋮	⋮	⋮
23	4	8855	0.174
24	4	10626	0.167
25	4	12650	0.160
⋮	⋮	⋮	⋮
38	3	8436	0.079
39	3	9139	0.077
40	3	9880	0.075
⋮	⋮	⋮	⋮
127	2	8001	0.016
128	2	8128	0.016
129	2	8256	0.016
⋮	⋮	⋮	⋮

Tabelle 3.4. *Überdeckung von* 89^2 *Zeichen*

n	k	$\binom{n}{k}$	p
1	n.a.	n.a.	n.a.
⋮	⋮	⋮	⋮
22	11	705432	0.500
23	9	817190	0.391
24	8	735471	0.333
25	8	1081575	0.320
26	8	1562275	0.308
27	7	888030	0.259
28	7	1184040	0.250
29	7	1560780	0.241
⋮	⋮	⋮	⋮
31	6	736281	0.194
32	6	906192	0.188
33	6	1107568	0.182
⋮	⋮	⋮	⋮
41	5	749398	0.122
42	5	850668	0.119
43	5	962598	0.116
⋮	⋮	⋮	⋮
66	4	720720	0.061
67	4	766480	0.060
68	4	814385	0.059
⋮	⋮	⋮	⋮
163	3	708561	0.018
164	3	721764	0.018
165	3	735130	0.018
⋮	⋮	⋮	⋮
1188	2	705078	0.002
1189	2	706266	0.002
1190	2	707455	0.002
⋮	⋮	⋮	⋮

Tabelle 3.5. *Überdeckung von* 89^3 *Zeichen*

3.4. Zählfolgen

Es sollen nun Sequenzen auch von mehr als 4 Assoziationen in einen Matrixspeicher eingetragen werden.

Wir betrachten zunächst unabhängig vom Problem der Datenspeicherung die Situation, statt *unabhängiger* Assoziationen $F^t \mapsto A^t$, wie dies der allgemeinen Nutzung eines Assoziativspeichers entspricht, eine *Folge* von Assoziationen $F^t \mapsto A^t = F^{t+1}$ zu speichern, wie in (18) auf S. 23 angedeutet.

Idee: Unter Ausschöpfung möglichst des gesamten kombinatorischen Potentials wird deterministisch eine kanonische Schlüsselfolge für einzelne Datenzeilen verwendet.

Unter den gegebenen Randbedingungen beginnen wir der Vollständigkeit halber mit einem einfacheren Sonderfall, auch um die Darstellung zu erläutern:

Vermeidung von Indexüberschneidungen

Aufbauend auf den einfachsten Mengen von Codeworten in [Dierks 05] [(21)] fordern wir zunächst: $\forall s, t, s \neq t : F^s \cap F^t = \emptyset$.

Wenn alle F^s paarweise keine gemeinsamen Bits aufweisen, ergeben sich beim Einspeichern in einen leeren Assoziativspeicher keinerlei Überschneidungen.

Die Lernregeln (K) und (L) fallen damit zu *einer* Lernregel zusammen, denn ein Überlagerungseffekt tritt nicht auf.

Überschneidungsfreie Vektoren

Es lassen sich sich daraus genau $T := n/k = 5$ verschiedene Vektoren gewinnen. Da die Indexnummern der Bitpositionen – das bedeutet die Zeilen und Spalten der Assoziativspeicher-Matrix – beliebig permutiert werden können [(22)] , ohne irgendwelche Struktureigenschaften zu verändern, wählen wir aus den isomorphen Permutationen – von denen hier $5! = 120$ existieren – der Indizes jene, welche eine kanonische Liste der Vektoren V^t ergibt [(23)] :

[(21)] Dort gilt für Schlüsselwerte in den *besonderen* Speichermatrizen A und K: „Beim Kodieren der Programmzeilen und Variablen [...] beachtet [...] die Orthogonalitätsbedingung. Das bedeutet, dass je zwei Variablen [...] keine Eins gemeinsam am selben Platz besitzen[...]“ (S. 90),

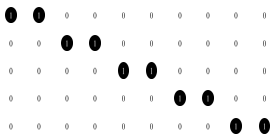
[(22)] ähnlich [Dierks 05] Abb. 67 auf S. 89, dort im Bereich *Variable*, dort allerdings mit 3 gesetzten 1-en und in anderer Anordnung der Spalten

[(23)] Diese Auswahl ist *hier* ohne Belang. Es lassen sich jedoch durch Permutation dieser Adressfolgen später immer Speichervarianten angeben, was

- Auswirkungen auf die Speicherkapazität und mögliche Symbolerkennungsleistung haben könnte und
- eine Rekonstruktion eines Textes unter Umständen erschweren kann, falls nur Teile der geplanterweise dazu benötigten weiteren Informationen zur Verfügung stehen

$t = 0, \ldots, T-1 \mapsto V^t$
$0 \mapsto \%1100000000$
$1 \mapsto \%0011000000$
$2 \mapsto \%0000110000$
$3 \mapsto \%0000001100$
$4 \mapsto \%0000000011$

Dies wollen wir auch so darstellen:



Als erste Idee zur Nutzung versucht man die 4 Schritte lange Assoziationsfolge

$$V^0 \mapsto V^1 \mapsto V^2 \mapsto V^3 \mapsto V^4$$

in den Assoziativspeicher einzuschreiben. Die 4 einzelnen Assoziationen lauten also (Abbildung 3.8)

Assoziationsfolge

$\{(V^0, V^1),$
$(V^1, V^2),$
$(V^2, V^3),$
$(V^3, V^4)\}$

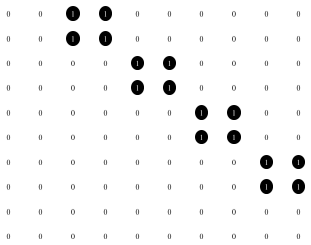
bzw.

Abbildung 3.8. *Versetzte überlappende Paarung einer Vektorfolge: Assoziationskette*

– wobei hier innerhalb einer Zeile immer links die F^t und rechts daneben die gewünschten A^t gelistet sind.

Das Eintragen gemäss der *Lernregel (L)* ergibt diese Belegung der 10×10-Matrix:

mit Assoziationsfolge belegter Assoziativspeicher



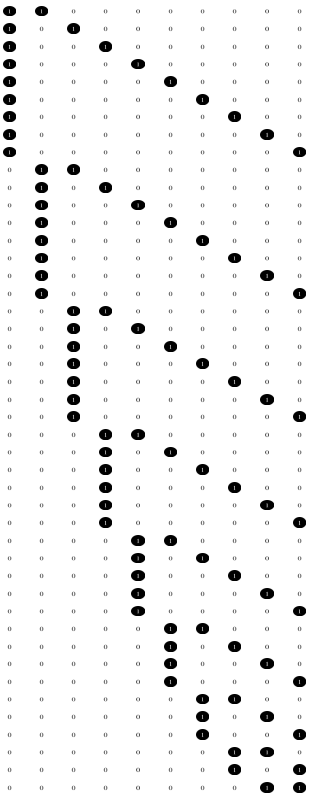
Es wäre möglich, auch die Assoziation (V^4, V^0) noch einzuspeichern, ohne Überlappungen zu verursachen. Mehr als $T-1 = 5$ Assoziationen in das Memory zu schreiben, ist jedoch so nicht möglich; und die Speicherausnutzung dieses Verfahrens ist natürlich äusserst ungünstig, da von jedem Vektor genau k redundante Kopien vorliegen.

Überlappende Indexkombinationen

Im allgemeinen wird man daher von dieser nicht-überlappenden Kodierung keinen Gebrauch machen, sondern beliebige Vektoren mit 2 gesetzten Index-Bits zulassen [24]

vollständige Liste

Die vollständige Liste der $\binom{10}{2}$ möglichen Vektoren sind alle 2-elementigen Auswahlen der Indexelemente $\{0, \ldots, 9\}$, wie in der Liste zu sehen:



Auch diese Liste ist wieder im Sinne einer kanonischen Ordnung sortiert [25] .

[24] Diese Möglichkeit wurde in [Dierks 05] S. 90 für einen deterministischen Ablauf in den beiden Speichermatrizen A und K ausdrücklich ausgeschlossen und ist erst für die später in [Bentz, Dierks 12] definierte *&afolge* in der L-Matrix erlaubt

[25] Wenn man die Zeilen von unten nach oben liest und als Binärdarstellungen von natürlichen Zahlen interpretiert, erhält man gerade die natürliche Reihenfolge. Wir bevorzugen hier

Während sich hier bei $\binom{n}{k} = \binom{10}{2}$ Möglichkeiten die komplette Sequenz noch bequem notieren lässt, ist dies z.B. bei $\binom{4000}{5} = 8512018660000800$ Indexkombinationen – wie sie später Verwendung finden – nicht mehr der Fall. Wenn aus dieser Fülle eine bestimmte Zeile gefragt ist, lässt sich die Indexmenge $I_{i,n,k} = \{b_0, \ldots, b_k\} \subset \mathcal{N}$ zum Vektor V^i der Zeile i relativ einfach [(26)] berechnen:

Index der linkesten 1:

$$b_0 = \min\{b \in \mathcal{N} : \sum_{a=0}^{b} \binom{n-1-a}{k-1} > i\}$$

und daraus:

$$I_{i,n,k} = \{b_0\} \cup \{j + b + 1 : j \in I_{i-s(b),n-1-b,k-1}\}$$

wobei $Start_b$:

$$s(b) = \min\{j : \forall k < b : V^i_j = 0\}$$

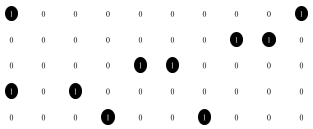
die Zeile ist, wo zum ersten Mal in der Folge die Stelle b gerade das linkeste Bit ist.

Zum direkten Vergleich mit dem vorigen Bild wollen wir ebenfalls einmal 5 Vektoren auswählen. Es bietet sich hier nun nicht an, direkt die ersten Vektoren aus der Liste zu verwenden.

Auch die letzten Vektoren nutzen nicht einmal eine Breite von 5 aus. Etwa jeden zehnten Vektor zu nehmen, würde wegen der Sortierung eine unerwünschte Präferenz der linken Seite bedeuten.

Wir treffen demnach eine (deterministisch pseudo-)zufällige Auswahl von 5 Vektoren: zufällige Auswahl

$\{V^0,$
$V^1,$
$V^2,$ $=$
$V^3,$
$V^4\}$



Die zugehörigen Assoziationen/Lernpaare sind dann wieder durch eine versetzte Paarung mit sich selbst gegeben:

---

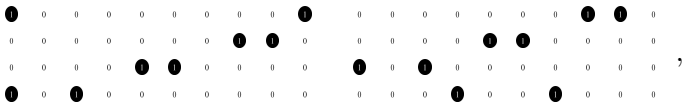
jedoch diese Reihenfolge, weil dann im Matrixspeicher bei nur teilweiser Verwendung der Liste die meisten Bits sich einfach sichtbar oben links befinden, wo wir es auch gewohnt sind, mit dem Schreiben zu beginnen.

[(26)] für relativ kleine k „nahe bei“ $O(\log n)$, vgl. http://www.thelowlyprogrammer.com/2010/04/indexing-and-enumerating-subsets-of.html

Assoziationsfolge

$$\{(V^0, V^1), (V^1, V^2), (V^2, V^3), (V^3, V^4)\}$$

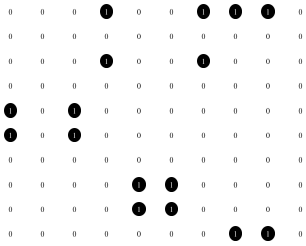
bzw.



,

gelernte Zufallsfolge

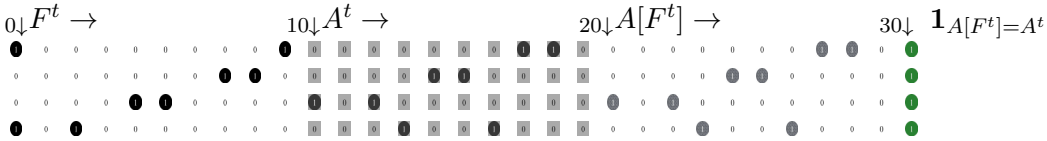
was zu der folgenden Speichermatrix führt:



Während hier die Vektoren nicht mehr disjunkt bezüglich der genutzten Indexpositionen sind, und es daher schon Überlagerungen im Speicher gibt (Zeile 0 hat mehr als zwei gesetzte Bits), führt dies noch zu keinen Ausleseproblemen:

Speicherprüfung

V^0 und V^4 können noch mit korrektem Ergebnis angefragt werden [27] :



Im Bild ist der Kontrolllauf des Speichers zu verfolgen, in dem die $T = 4$ Assoziationen der Reihe nach am Speicher als F^t wieder angefragt werden. Es sind die folgenden Positionen zu *einem* Vektor der Länge 31 zusammengefasst:

Bits	Symbol	Bedeutung
0−9	F^t	der jeweilige Fragevektor
10−19	A^t	der gelernte zugehörige Antwortvektor *(Soll)*
20−29	$A[F^t]$	der aus dem Memory erhaltene Antwortvektor *(Ist)*
30	$\mathbf{1}_{A[F^t]=A^t}$	Indikator für Übereinstimmung von *Soll* und *Ist*

Zur besseren Unterscheidbarkeit sind die originalen A^t grau hinterlegt.

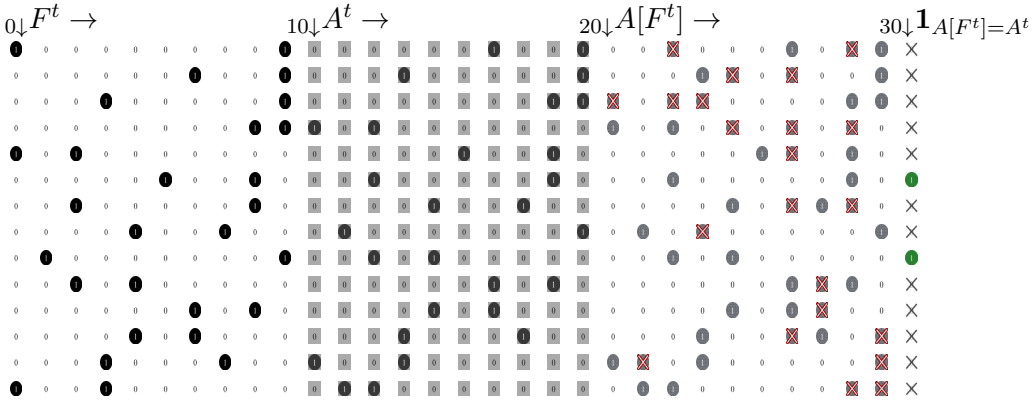
Überbelegung

Nach den eben dargestellten einfachsten Beispielen, die ohne Probleme fehlerfrei gespeichert und gelesen werden konnten, zeigen wir nun, wie sich ein solches System verhält, wenn *mehr* Assoziationen eingegeben werden.

[27] Es ist auch noch anzumerken, dass $k^2 \cdot T$ 1-Bits zur Erzeugung der Matrix verwendet wurden und auch genausoviele sich darin finden: Auf den Einzelpositionen gibt es noch keine streng lokalen Überlagerungseffekte.

Im sonst unveränderten Beispiel wählen wir nun zufällig $T = 15$ Vektoren V^t.

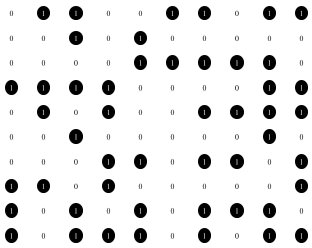
Speicherprüfung bei Überbelegung



Die fälschlich gelieferten 1-Bits bei der Abfrage sind im dritten Bereich der Liste durch rote Kreuze markiert.

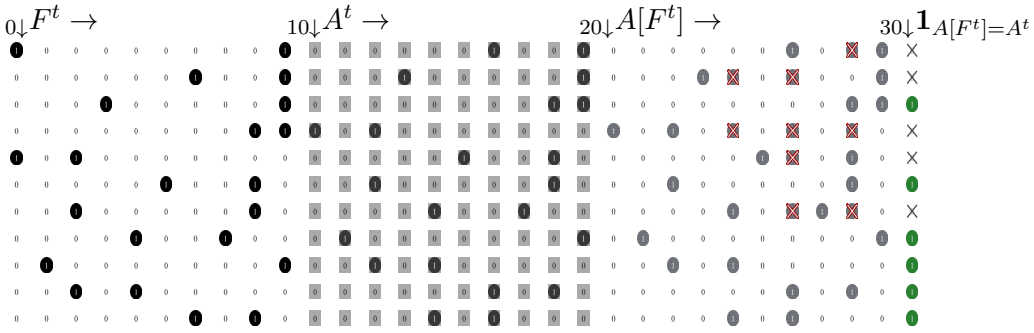
Die Überbelegung der Speichermatrix ist im Bild der Matrix selbst schon zu erahnen:

überbelegter Speicher



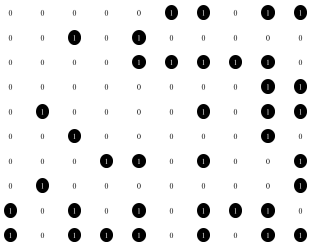
Versuchen wir es noch einmal mit $T = 12$ Vektoren:

2. Speicherprüfung bei Überbelegung



Hier werden (rechte Spalte) immerhin 6 von 11 Assoziationen korrekt gespeichert.

2. überbelegter Speicher



Wir sehen durch die Überlappungen der Bitpositionen in den Vektoren im Speicher nur 38 Bits gesetzt von den 48 durch die Assoziationen gegebenen Verknüpfungen.

3.4.1. Theoretische Fehlerwahrscheinlichkeiten

Wegen der geringen Spärlichkeit, entsprechend einem kleinen Alphabet, kann in diesen illustrativen Beispielen die maximale Ausnutzung des Assoziativspeichers nicht erreicht werden.

Ausserdem gilt: obwohl $P = 0.38 \leq 0.5 = P_{opt}$ (28) sind bei der Verwendung von Assoziativspeichern *immer* Fehler zu erwarten.

[Palm 80] erwähnt als (4.3) auf S. 25 (29) als Näherung für die Fehlerwahrscheinlichkeit p_f eines gegebenen Antwortbits, also die Wahrscheinlichkeit $p_{A_i:0\to1}$ einer fälschlich gelieferten 1 anstelle einer 0 beim Auslesen

$$p_{A_i:0\to1} \approx P^k = \frac{1}{2^k} \qquad (p_f)$$

für jede der $n - k$ Stellen. Die Wahrscheinlichkeit, einen Vektor fehlerlos zurückzuerhalten, ist infolge

$$(1 - p_f)^{n-k} = \left(1 - \frac{1}{2^k}\right)^{n-k} \qquad (p_+)$$

und ist für kleine k auch sehr klein (siehe Tabelle 3.8) für n in weiten Bereichen. Tabelle 3.9 zeigt die theoretischen Werte für andere (kleinere) P. Eingetragen ist dort immer auch das zugrundeliegende optimale T (Anzahl der Assoziationen) hier angenähert durch

$$T \approx \frac{n^2}{k\,ld\,n} ln\,2$$

in Analogie zu [Palm 80] (4.17).

n	k	P	p_+	$\approx T$
10	2	0.500	0.100	10
10	3	0.500	0.393	6
10	4	0.500	0.679	5
10	5	0.500	0.853	4
10	6	0.500	0.939	3
10	7	0.500	0.977	2
20	2	0.500	0.006	32
20	3	0.500	0.103	21
20	4	0.500	0.356	16
20	5	0.500	0.621	12
20	6	0.500	0.802	10
20	7	0.500	0.903	9
30	2	0.500	3.175e-04	63
30	3	0.500	0.027	42
30	4	0.500	0.187	31
30	5	0.500	0.452	25
30	6	0.500	0.685	21
30	7	0.500	0.835	18

Tabelle 3.6. *Kleine Matrizen: Wahrscheinlichkeit p_+ für Fehlerlosigkeit einer Assoziation bei optimalen Assoziationsanzahlen*

e vs.
chtung

Schliesslich stimmt der Wert aus Tabelle 3.6, wo einige Parameter wie im Bereich des Beispiels gelistet sind, mit unserem beobachteten $T - 1 = 11$ einigermassen überein, so dass wir nicht erwarten dürfen, noch mehr Datensätze einspeichern zu können.

Zu Tabelle 3.9 sei noch angemerkt, dass für kleinere P möglicherweise bessere Ergebnisse erzielt werden können. Hierzu schreibt [Palm 13]:

> „In these applications there already appeared some indications that the limit of ln 2 retrievable bits per hardware bit, i.e. the efficiency limit of ln 2 can be surpassed. [...] we found a regime of ultra-sparse memory patterns, where the storage matrix is also a sparse matrix and the [...] efficiency of the memory approaches 1“

Ob sich derartige Darstellungsformen für unsere Zwecke nutzen lassen, ist jetzt zu untersuchen – insbesondere in Hinblick auf die Darstellungseffizienz derartiger Matrizen in realen Computersystemen. Es ist jedoch nicht ersichtlich, warum die Ergebnisse von [Hagström 96] hier nicht gelten sollten.

(28) [Palm 80] für ein Optimum der gespeicherten Information pro Zelle $\frac{I}{n^2}$ (Definition der *information capacity*)

(29) allerdings nur asymptotisch für gut belegte Matrizen im Sinne der Parameter T, k

Symbol	bzw.	Bedeutung
$T-1$		Anzahl der Assoziationen
	T	Anzahl der Vektoren
n		Vektorlänge
k		Anzahl der in jedem Vektor gesetzten Bits
p	k/n	Belegungsdichte der Vektoren
p_+		Theoretische Wahrscheinlichkeit, ein einzelnes t korrekt auslesen zu können
P		Belegungsdichte der Matrix

Tabelle 3.7. *Legende u.a. zu Tabellen 3.6 bis 3.9*

n	k	P	p_+	$\approx T$
100	2	0.500	5.702e-13	521
100	4	0.500	0.002	260
100	6	0.500	0.228	173
100	8	0.500	0.698	130
100	10	0.500	0.916	104
100	12	0.500	0.979	86
100	14	0.500	0.995	74
500	2	0.500	6.033e-63	9663
500	4	0.500	1.252e-14	4831
500	6	0.500	4.181e-04	3221
500	8	0.500	0.146	2415
500	10	0.500	0.620	1932
500	12	0.500	0.888	1610
500	14	0.500	0.971	1380
1000	2	0.500	2.047e-125	34776
1000	4	0.500	1.212e-28	17388
1000	6	0.500	1.591e-07	11592
1000	8	0.500	0.021	8694
1000	10	0.500	0.380	6955
1000	12	0.500	0.786	5796
1000	14	0.500	0.942	4968
4000	2	0.500	0.000e+00	463419
4000	4	0.500	9.936e-113	231709
4000	6	0.500	4.823e-28	154473
4000	8	0.500	1.639e-07	115854
4000	10	0.500	0.020	92683
4000	12	0.500	0.378	77236
4000	14	0.500	0.784	66202
10000	2	0.500	0.000e+00	2608226
10000	4	0.500	6.682e-281	1304113
10000	6	0.500	4.434e-69	869408
10000	8	0.500	1.037e-17	652056
10000	10	0.500	5.768e-05	521645
10000	12	0.500	0.087	434704
10000	14	0.500	0.544	372603

Tabelle 3.8. *$P = \frac{1}{2}$: Wahrscheinlichkeit p_+ für Fehlerlosigkeit einer Assoziation bei optimalen Assoziationsanzahlen*

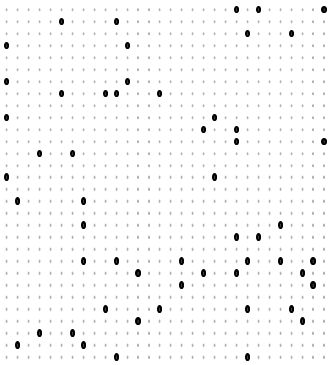
n	k	P	p_+	$\approx T$
100	2	0.250	0.002	521
100	5	0.250	0.911	208
100	8	0.250	0.999	130
100	11	0.250	1.000	94
100	14	0.250	1.000	74
1000	2	0.250	1.065e-28	34776
1000	5	0.250	0.378	13910
1000	8	0.250	0.985	8694
1000	11	0.250	1.000	6322
1000	14	0.250	1.000	4968
10000	2	0.250	5.872e-281	2608226
10000	5	0.250	5.740e-05	1043290
10000	8	0.250	0.859	652056
10000	11	0.250	0.998	474222
10000	14	0.250	1.000	372603
100	2	0.100	0.373	521
100	5	0.100	0.999	208
100	8	0.100	1.000	130
100	11	0.100	1.000	94
100	14	0.100	1.000	74
1000	2	0.100	4.405e-05	34776
1000	5	0.100	0.990	13910
1000	8	0.100	1.000	8694
1000	11	0.100	1.000	6322
1000	14	0.100	1.000	4968
10000	2	0.100	2.294e-44	2608226
10000	5	0.100	0.905	1043290
10000	8	0.100	1.000	652056
10000	11	0.100	1.000	474222
10000	14	0.100	1.000	372603

Tabelle 3.9. *$P < \frac{1}{2}$: Wahrscheinlichkeit p_+ für Fehlerlosigkeit einer Assoziation bei optimalen Assoziationsanzahlen*

3.4.2. Korrektur durch Spekulation

14 Assoziationen zwischen 15 von $\binom{30}{3}$ Vektoren

Wir wählen jetzt $n = 30$ und $k = 2$, um etwas mehr Daten einspeichern zu können. $T - 1 = 14$ zufällig gewählte Assoziationen können einmal korrekt gespeichert werden:

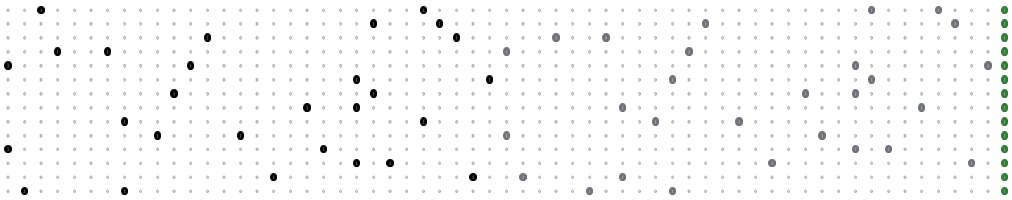


30-2-14 Matrix

Wir verzichten im folgenden bei diesen grösseren n auf die Darstellung der Spalte für A^t. Der jeweilige A^t kann bei diesem Sequenz-Lernen ja immer als F^{t+1} direkt in der Folgezeile abgelesen werden und erscheint auch rechts als Ergebnis $A[F^t]$, wenn man die ja immer extra markierten fehlerhaften 1-Bits ignoriert.

$F^t.A[F^t].1_{A[F^t]=A^t}$ soll die Aneinanderreihung der Teilvektoren in *einem* Bild, d.i. in jeder Zeile darin, bezeichnen.

Der Teil $A[F^t]$ ist etwas schwächer (*grau*) dargestellt, die Korrektheit der Assoziationen ganz rechts ist *grün* bzw. im Fehlerfalle durch ein $\times$ markiert.

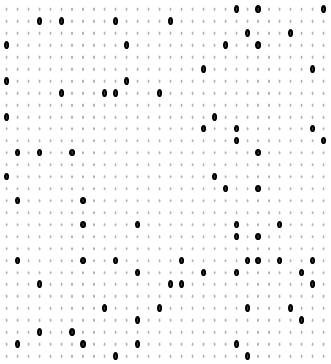


30-2-14 Prüfung, $F^t.A[F^t].1_{A[F^t]=A^t}$ dargestellt

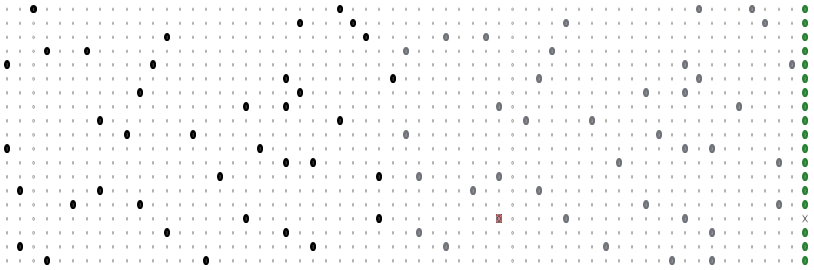
Hier finden sich also bei einem fehlerlosen Speicher- und Ausleseprozess keine markierten Fehlerbits, und die Korrektheit jeder Zeile ist rechts sichtbar.

19 Assoziationen zwischen 20 von $\binom{30}{2}$ Vektoren

Die Auswahl von $T-1=19$ Assoziationen ergibt nun *einen Fehler* (rechte Spalte der Prüfung in Zeile $t=15$ nicht gesetzt):



30-2-19 Matrix



einfacher Fehler

30-2-19 Prüfung

Wir wollen sehen, ob sich dieser Fehler aber beim Auslesen korrigieren liesse: In diesem Falle wurde *genau ein* zusätzliches (fehlerhaft) gesetztes Bit im Ausgabevektor $A[F^{15}] \neq A^{15} = F^{16}$ geliefert, also 3 Bits statt 2, wie in allen F^t, A^t.

Danach ist dann jedoch wieder $A[F^{16}] = A^{16}$ korrekt. Allerdings ist F^{16} bei einem *seriellen Auslesen* ja noch nicht bekannt.

Es lohnt aber zumindest den Versuch, alle 3 zumindest der Dichte nach korrekten Kandidaten, welche einfach aus $A[F^{15}]$ durch Weglassen eines Bits bestimmt werden können, ob sie bei einer Anfrage an den Speicher einen der Dichte nach gültigen Vektor liefern [30] , genauer: *genau einer der Vektoren* (hier zufällig der letzte) eine Antwort mit genau 2 gesetzten Bits ergibt.

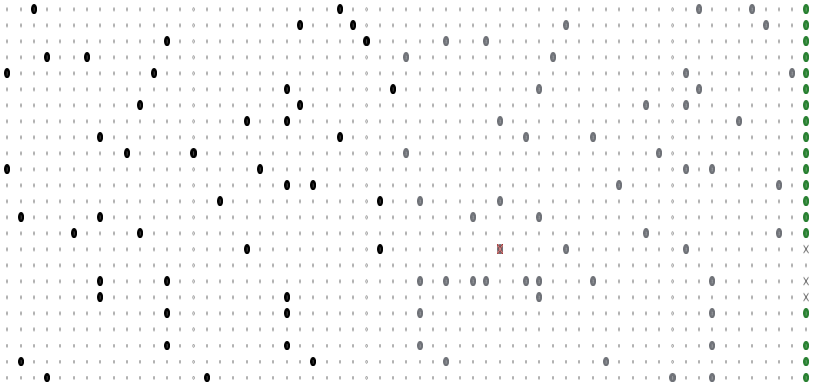
Sei hier $f = \left|A[F^t]\right| - \underbrace{\left|A^t\right|}_{k}$ die Anzahl der fehlerhaft gelieferten 1-Bits einer Anfrage $A[F^t]$, hier also $f=1$. Zur Darstellung ist bei dem aufgetretenen Auslesefehler jetzt eine Freizeile eingefügt, worauf die $\binom{k+f}{k}$ als korrektes Ergebnis infragekommenden Indexuntermengen des $A[F^t]$ als $F^{(t+1).i}$ dann geprüft werden. In diesem Zeilenblock gilt: Die letzte Spalte ist genau dann gleich 1 dar-

[30] Es ist anzumerken, dass eine Hardware-Realisierung der Assoziativmatrix hierdurch erheblich verkompliziert würde, da mehrere Register und eine Art von Mikroprogramm für das Ausprobieren der Möglichkeiten benötigt werden

gestellt, wenn $\left|A[F^{(t+1).i}]\right| = k$.

Wenn dieses letzte Bit *in genau einer Zeile* des Blocks der spekulativen $F^{(t+1).i}$ gesetzt ist, heisst dies, dass der Fehler einfach und mit Gewissheit korrigiert werden kann. [31]

Auf eine weitere Freizeile folgt dann die Fortsetzung der normalen Abfragen der weiteren F^t [32] .



30-2-19 Prüfung mit Korrektur

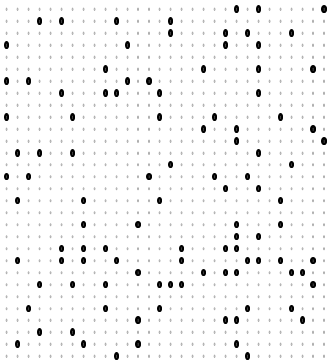
Fehlerkorrektur möglich

Wir sehen hier im Ablauf, wie nach Erhalt eines falschen $A[F^t]$ mit *einem* zusätzlichen Bit alle (hier $i \in \{0, \ldots, \binom{3}{2} - 1\}$) Teilmengen mit geeigneter Bitdichte geprüft werden.

Und *tatsächlich* erfüllt nur *einer* der spekulativ geprüften Vektoren (links als $F^{(t+1).i}$) dann $\left|A[F^{(t+1).i}]\right| = 2$ und ergibt damit einen gültigen Vektor. Es kann in diesem Beispiel also durch Einbeziehung von *Wissen über die verwendete Kodierung* die korrekte Folge trotz der Störung wieder ausgelesen werden.

29 Assoziationen zwischen 30 von $\binom{30}{2}$ Vektoren

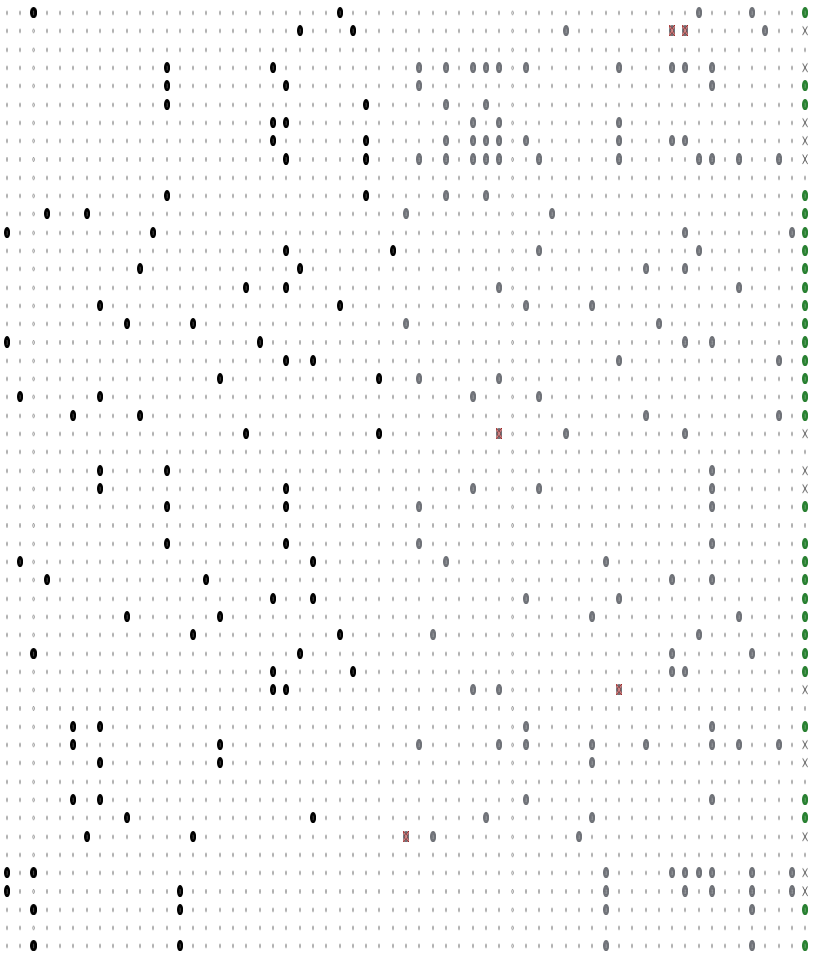
$T - 1 = 29$ ergibt schon *sieben* Fehler:



30-2-29 Matrix

[31] Diese Bedingung liesse sich noch abschwächen auf *genau ein* $\left|A[F^{(t+1).i}]\right| \geq k$

[32] hier immer so dargestellt, auch wenn wie beschrieben keine eindeutige Korrektur möglich gewesen ist



30-2-29 Prüfung

3 von 4 Fehlern können einfach korrigiert werden

Der bekannte Fehler bei $t = 15$ sowie die zwei nachfolgenden Fehler (in zusätzlichen Assoziationen) lassen sich wieder leicht korrigieren. Bei der zuletzt noch korrekt ausgelesenen Assoziation $t = 1$ dagegen tritt jetzt ein *neuer* Lesefehler, resultierend aus den später eingespeicherten Daten, auf.

Hier ist allerdings eine Wiederherstellbarkeit mit der Vorausschau um nur einen weiteren Schritt nicht gegeben: Von den hier $\binom{4}{2}$ zu prüfenden Varianten ergeben *zwei* einen gültig belegten potentiellen Folgevektor A^1.

Auch wenn (nicht im Beispiel) mehrere Fehler in direkt aufeinanderfolgenden Zeilen auftreten, lässt sich nur mit einer tiefergehenden Baumsuche über alle möglichen korrekten Sequenzen ein solcher Fehler korrigieren, was wir jedoch jetzt nicht weiter betrachten möchten.

3.4.3. Einfache Textspeicherung

Unter Verwendung geeigneter, der im letzten Abschnitt definierten $n-k$-Abfolgen, von denen wir eine deterministisch pseudo-zufällige Auswahl treffen als F^t, wollen wir nun beginnen, Texte in einen Assoziativspeicher einzuschreiben. Als Parameter wählen wir für ein einfaches Beispiel:

Symbol	Wert	Bedeutung
(V^t)	s.u.	„Worte“ des Textes
$T = \lvert\{\tau_t\}\rvert = \lvert\{V^t\}\rvert$	5	Anzahl der Assoziationen
$n = b \cdot c$	30	Matrixgrösse ist $n \times n$
k	2	Anzahl der in jedem Vektor F^t gesetzten Bits
p_F	k/n	Belegungsdichte der Fragevektoren, hier also der „Adressen“
p_A	k/n	Belegungsdichte der Antwortvektoren, hier also der „Textblöcke“
P		(resultierende) Belegungsdichte der Matrix
C	{A,B,C,D,E,F}	Alphabet des Textes
c	5	Anzahl der Zeichen pro Zeile
$b = \lvert C\rvert$	6	Umfang des Alphabets, damit Basis des $CIAP$
(A^t)	$CIAP[V^t, b]$	Folge der A^t: jeweiliges V^t kodiert als „Buchstabe auf Platz“
t	$\in \{0, \ldots, T-1\}$	Folgenummer der Assoziation
(F^t)	$RS[n, k, s]$	Folge der F^t: (reproduzierbare) pseudo-zufällige Auswahl, siehe linker Teil von Abbildung 3.9 und dann folgende
s	42	Startwert des Zufallsgenerators

Einfacher Beispieltext

Als Beispieltext verwenden wir:

```
aaaaa abcde bcdef fffff acedf
```

Buchstabe-auf-Platz-Kodierung

Wir verwenden eine konstante Blockgrösse von c Zeichen aus dem Alphabet mit Umfang b. Bei einer Kodierung „Zeichen auf Position“, wobei die Positionen $\{0, \ldots, c-1\}$ benannt sind, bedeuten also die Bits $0, \ldots, 29$ der Reihe nach:

0: <**A**> an Position 0,
1: <**B**> an Position 0,
2: <**C**> an Position 0,
3: <**D**> an Position 0,
4: <**E**> an Position 0,
5: <**F**> an Position 0,
6: <**A**> an Position 1,
7: <**B**> an Position 1,
8: <**C**> an Position 1,
... bis hin zu
28: <**E**> an Position 4,
29: <**F**> an Position 4.

Es ist hier zu bemerken:

- Jeder Block $\{[i,\ldots,i+b-1] : i \in \{0,b,2b,\ldots\}\}$ repräsentiert dabei genau eine der c Stellen im Klartext. Insbesondere ist daher in jedem dieser Blöcke *genau ein* Bit gesetzt, denn auf jeder Klartextstelle findet sich *ein* Zeichen [(33)] .
- Daraus ergibt sich: Diese Kodierung ist eine Bijektion und damit die Dekodierung d, wodurch sich der Text später aus dem Speicher rekonstruieren lässt.

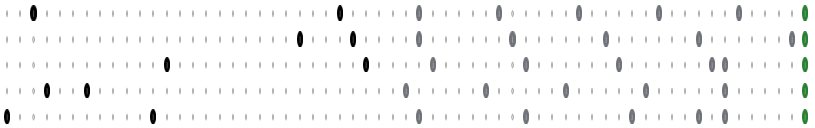


Abbildung 3.9. Prüfung eines einfachen Textes von 5 Zeilenblöcken adressiert durch eine kombinatorische Pseudo-Zufallssequenz

Prüfung der Speicherung eines einfachen $|\{\tau_t\}| = 5$

Die korrekte Auslesbarkeit ist nun nicht verwunderlich: Wie in der linken Hälfte des Bildes oben zu sehen, gibt es (hier zufällig) noch keine Überlappungen bei der Einspeicherung. Die 5 kodierten Textzeilen A^t finden sich jeweils doppelt in 10 der 30 Speicherzeilen unverändert aufbewahrt, was dem anfangs unter „Vermeidung von Indexüberschneidungen“ Aufgezeigten direkt entspricht.
Erhöhen wir also die Anzahl der Textzeilen auf 10 (Abb. 3.10):

Abbildung 3.10. Prüfung von 10 Zeilen mit Pseudo-Zufallssequenz

Prüfung der Speicherung eines einfachen $|\{\tau_t\}| = 10$

Hier gibt es bereits Überschneidungen im unteren Zeilendrittel des Speichers, erkennbar an den mehr als c gesetzten Bits in einer Zeile in Abbildung 3.11.

[(33)] Diese Eigenschaft kann auch zur Fehlerkorrektur beim Auslesen des Speichers herangezogen werden, um bei der Identifikation fälschlich gesetzter Bits die Anzahl der Möglichkeiten *deutlich* zu reduzieren.

$|\{\tau_t\}| = 10$

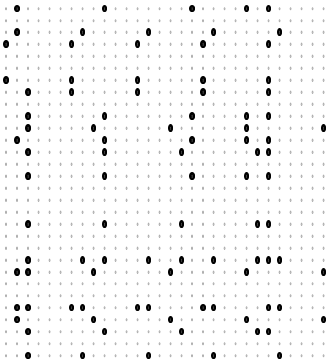


Abbildung 3.11. *Textspeicher mit Überlappungen bei* 10 *Zeilenblöcken*

Für 20 Assoziationen wird als Testtext jetzt die ergänzte Blockfolge

```
aaaaa abcde bcdef fffff acedf baaaa bbcde bcdef bffff bcedf
caaaa cbcde ccdef cffff ccedf daaaa dbcde dcdef dffff dcedf
```

verwendet (Abbildung 3.12): [34]

20 Blöcke zu 5 Zeichen

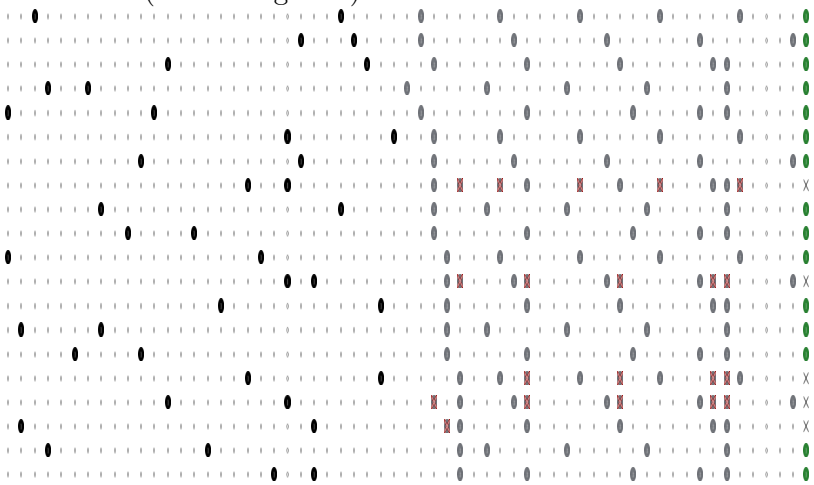


Abbildung 3.12. Prüfung von 20 Zeilen mit Pseudo-Zufallssequenz

Prüfung der Speicherung eines einfachen $|\{\tau_t\}| = 20$

Es treten hier bereits Überlagerungen auf, welche zu relativ vielen Fehlern führen. Wie bereits angemerkt, kann die Anzahl der zu prüfenden $A^{t.i}$ im Falle des A^{16} zwar von $\binom{10}{5} = 252$ auf $2^5 = 32$ reduziert werden, sowie bei A^{17} von $\binom{6}{5} = 6$ auf $2^1 = 2$, indem statt allein der Zieldichte auch die Blockbildung bezüglich der Zeichenpositionen genutzt wird, jedoch ist hier das Problem:
Die zu prüfenden $A^{t.i}$ sind *überhaupt nicht als* $F^{t+1.i}$ *geeignet*, da gar keine Sequenz in den Assoziativspeicher eingeschrieben wurde. Man vergleiche in der anfangs in diesem Abschnitt gegebenen Tabelle der Parameter – wo im wei-

[34] Zugegeben ist dieser Text „degeneriert“, da – vom ersten Zeichen einmal abgesehen – jeder Blocktext 4× erscheint.

teren nur (V^t) und damit T vergrössert wurden – dass die F^t gar nicht von irgendwelchen V^t abhängig sind.

Natürlich Sequenzen!

Neben dem auf Seite 13 vorgebrachten eher psychologischen Argument bezüglich des Antwortumfanges $|A^t|$, findet sich hier eine Rechtfertigung für die Idee, $|A^t| = |F^t|$ und genauer möglichst sogar dieselbe Kodierung zu verwenden.
Eine entsprechende Speicherung des Textes als Sequenz wird im Abschnitt 3.5 abgehandelt; zunächst betrachten wir jedoch noch Varianten der bisher verwendeten Struktur zur Speicherung.

Varianten

Man findet schnell die folgenden Möglichkeiten der Variation:

- Änderung von Parametern, insbesondere
  - n,k (typischerweise Verringerung des $p_F = \frac{n}{k}$ und Vergrösserung von n) der F^t durch Wahl anderer Adressfolgen, oder
  - direkte Verringerung von p_A, was nur durch Wahl eines anderen Alphabets möglich ist und somit automatisch zu Tupelkodierungen über mehr als eine Stelle des Urtextes führt [35] .
- Modifikation des Verfahrens, indem Auslesefehler beim Einspeichern detektiert und in Folge entweder
  - die A^t modifiziert werden z.B. durch Änderung der Blockbildung, möglicherweise auch auf Blöcke variabler Länge oder
  - *andere Korrekturmechanismen* in dem Sinne, dass auch die F^t beim Erkennen eines Fehlers sowohl beim Einspeichern als auch beim Auslesen *ergänzt* oder *ersetzt* werden [36] .

[35] ...dies ist im obigen Beispiel eigentlich zwingend, da $p_A = \frac{1}{6}$ definitiv nicht im optimalen Bereich eines Assoziativspeichers operiert. Selbst bei einem „normalen“ Alphabet von 26 oder mit Varianten und Sonderzeichen von etwa 100 ist dies ja noch zu empfehlen (vgl. Abschnitt 3.3.2 und Abschnitt 3.6.1)

[36] letzteres entspricht einer typischen Kollisionsbehandlung bei Hashtabellen. Im Abschnitt 3.5.1 ist das entsprechende Verfahren für sequentielle Assoziationsbildung ausgeführt, wo es als Transformation l des Textes (V^t) dann sowohl die F^t als auch die A^t modifiziert

Parameterveränderung p_F: Als einfachste Variante prüfen wir $n = 30$, $k = 3$ (Abbildung 3.13). Variiertes p_F

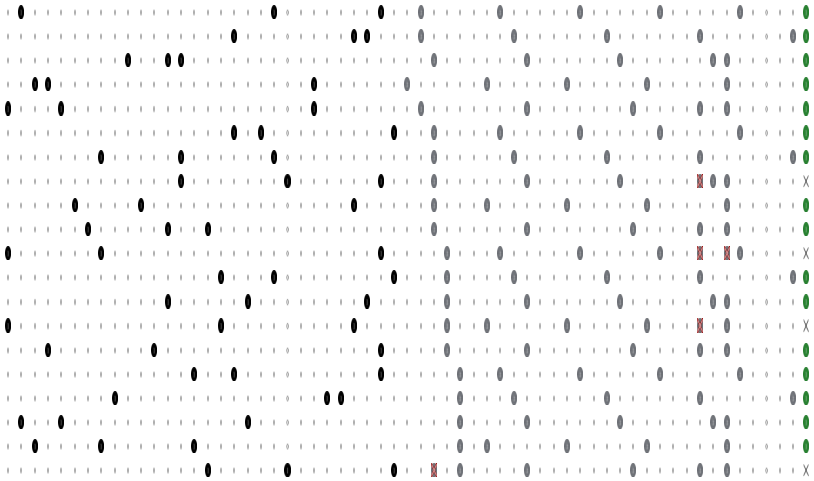


Abbildung 3.13. *20 Zeilenblöcke von* $\binom{30}{3}$ *Adressen*

Während die Fehler eigentlich aus den Überlagerungen stammen, hier jedoch sogar alle Datensätze durch das vergrösserte $p_F = \frac{1}{15} \rightarrow \frac{1}{10}$ *noch häufiger* eingetragen werden und dadurch die Belegungsdichte P des Speichers zunimmt, ist hier einmal eine der wesentlichen Eigenarten der Assoziativspeicher zu bemerken: Solange die Speicherdichte P des Matrixspeichers unterhalb des Optimums liegt, kann das Ergebnis (hier die Anzahl der korrekt abrufbaren Blöcke $recall_T = \frac{15}{20} = 0.75 \rightarrow \frac{16}{20} = 0.8$) durch Nutzung von Redundanzen verbessert werden . Qualitätsverbesserung durch Redundanz

Es ist aber auch noch besonders zu bemerken, dass [37] die Qualität (im Sinne des Durchschnitts m_f der Zahl falscher Bits) *innerhalb* der „falschen“ Resultate sich offenbar deutlich verbessert.

Man kann sagen, dass hier die Verbesserung aufgrund der Kombinatorik gegenüber der Verschlechterung aufgrund der Kollisionen noch überwiegt: Tabelle 3.6 zeigt bei $n = 30$ im Übergang von $k = 2 \rightarrow 3$ eine Vergrösserung der Erwartung einer korrekt zu rekonstruierenden Assoziation. Die konkreten Werte stimmen bei diesem geringen $T = 20$ jedoch offenbar noch nicht mit denen von T_{opt} überein.

Zum Vergleich noch dasselbe Experiment mit $k = 4$ (Abbildung 3.14). noch grösseres p_F

Hier stellt sich jetzt keine Verbesserung mehr ein, im Gegenteil, gegenüber $k = 3$ ist das Ergebnis wieder schlechter. Die Änderung bewegt sich zwar noch im Rahmen zufälliger Schwankungen, jedoch ist davon auszugehen, dass in dem Masse, wie wir uns mit P dem $P_{opt} = \frac{1}{2}$ annähern, wir auch damit rechnen müssen, dass die tabellierten p_+ angenommen werden. $p \approx \frac{2}{3} \cdot p_{opt}$?

[37] vgl. (p_f) auf S. 36

Abbildung 3.14. *20 Zeilenblöcke von* $\binom{30}{4}$ *Adressen*

In Abbildung 3.15 wird dieser Punkt langsam erreicht. Dies ist zwar optimal im Sinne der *information capacity* [38] , allerdings auch ungünstig in dem Sinne, dass bald nahezu jeder Vektor – wenn auch nur leicht – verfälscht ist.

Zufall?

Unklar ist aber dennoch, ob die gefundenen Fehlerzahlen eventuell doch nur einer ungünstigen oder speziellen Auswahl von (F^t) geschuldet sind bzw. andererseits dem sowohl weltfremden als auch andererseits hochgradig systematisch erzeugten Beispieltext.

Um mit grösserer Gewissheit festzustellen, wo die geringste Fehlerzahl – eventuell bei optimaler Ausnutzung – erreicht wird, führen wir das beschriebene Experiment *häufiger* aus.

Parameter

Wir verwenden dabei die folgenden Parameterwerte

Symbol	Wertebereich	Bedeutung
T		Anzahl der Assoziationen
n		Vektorlänge
k		Anzahl der in jedem Schlüssel-Vektor F^t gesetzten Bits
s	55	Startwert des Zufallsgenerators
N	20	Anzahl der Durchführungen des Experimentes [39]

[38] wie beschrieben in [Palm 13] bzw. [Palm 80], bzw. vgl. Def. auf S. 37

[39] wenn $N > 1$, so gilt s beim den ersten Lauf

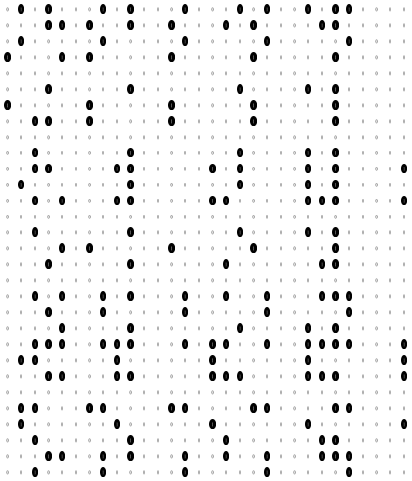


Abbildung 3.15. *Speichermatrix zu Abbildung 3.14*

beziehungsweise ermitteln diese Kenngrössen: Messwerte

Symbol	Wert	Bedeutung
p_F	k/n	Belegungsdichte der Adressvektoren
p_A	k'/n	Belegungsdichte der „kodierten Textblöcke“
P oder $p(a_{ij})$		Belegungsdichte der Matrix
e	$= \big\lvert\{t : A[F^t] \neq A^t\}\big\rvert$	Anzahl falscher Auslesevorgänge

Wir verwenden statt des bisherigen Beispieltextes jetzt Zufallstexte über demselben Alphabet, daher bezeichnen wir mit Texttypen

Symbol	Wert	Bedeutung
G		Typ des Experimental-Textgenerators, d.h. $V^t \sim G(t)$:
	$\mathcal{C}$	*Const*: vorgegebener Text
	$\mathcal{E}$	*Entropie*: Rauschen eines Pseudo-Zufallsgenerators

In Tabelle 3.10 sind die durchschnittlichen Fehlerzahlen m_e jetzt eines Zufalls-Textes über dem Alphabet <**ABCDEF**> gemittelt über jeweils $N = 20$ Versuche für $k = 2, \ldots, 6$ gesetzte Bits von $n = 30$ möglichen dargestellt.

T	2	3	4	5	6
7	0.35	0.50	0.60	0.10	0.40
9	0.75	1.05	0.75	1.60	1.00
11	1.40	1.60	1.45	2.05	1.65
13	2.55	3.05	3.05	2.45	2.65
15	3.50	4.55	3.85	3.90	4.40
17	5.40	4.80	5.00	5.65	5.15
19	7.40	7.40	6.70	7.75	6.80
21	9.30	9.65	8.65	9.50	9.95
23	10.85	11.60	11.90	11.30	10.75
25	12.45	13.10	12.45	13.45	12.70
27	14.50	14.75	14.20	14.35	14.90
29	17.35	17.10	16.55	16.75	17.45

Tabelle 3.10. *Zufällige Zeichen: Durchschnittliche Fehlerzahlen m_e über 20 Versuche, Spalten k=2,...*

Experimentelle Fehleranzahlen bei $p_A = \frac{1}{6}$

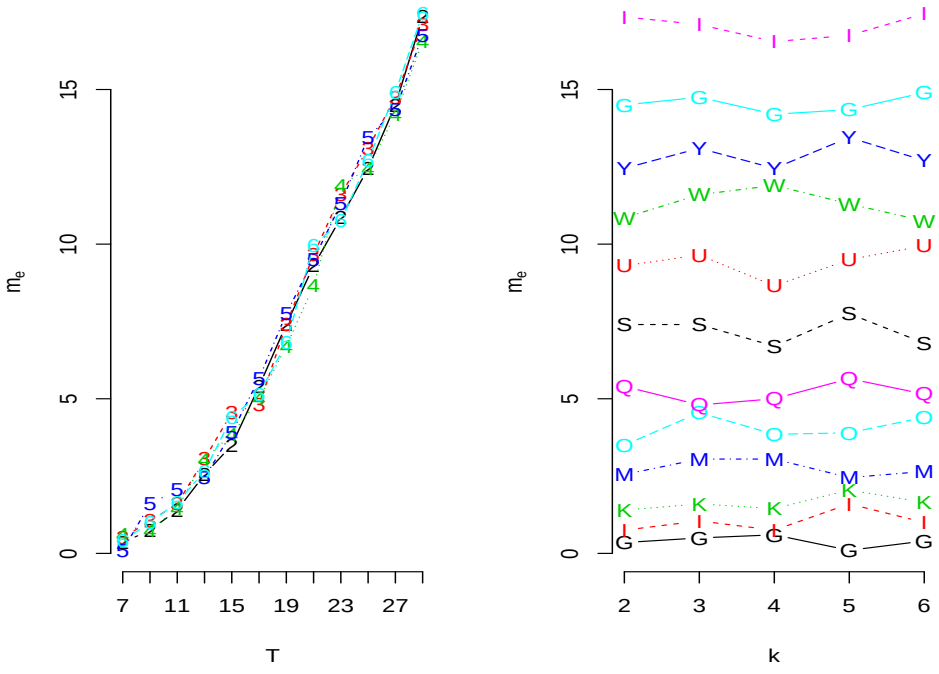


Abbildung 3.16. *Zufällige Zeichen: Durchschnittliche Fehlerzahlen m_e über 20 Versuche, Reihen für k = ... und T = ...; G,I,...,Y,G,I entsprechen dabei T=7,9,...,27,29*

Im Vergleich mit den Ergebnissen aus Abbildung 3.9-3.14 ist hier festzustellen, dass wider Erwarten *mehr* Fehler auftreten, wenn „Rauschen“ statt systematischer Texte eingespeichert wird (≈ 8 vs. 4 in Abb.3.13, ≈ 7 vs. 5 in Abb.3.14). Wir verstehen dies so, dass die inneren Ähnlichkeiten der systematischen Texte im Memory eher zusammenfallen, wodurch der gegebene Raum besser ausgenutzt werden kann. Offensichtlich findet tatsächlich auch automatisch eine Art von Komprimierung der Daten aufgrund der Gleichartigkeit statt.
Insgesamt sind die Ergebnisse jedenfalls nicht besonders überzeugend, was an der hohen Belegungsdichte der Vektoren beziehungsweise der kleinen Matrix liegt. Die Fehlerzahlen steigen im Einklang mit Tabelle 3.8, allerdings scheint k in diesem Bereich keinen besonderen Einfluss zu haben. Die Belegungsdichten mit $0.06 < p_F < 0.20$ und $p_A \approx 0.17$ liegen noch *beide* weit oberhalb von $p_s \leq 0.001$, was wir im allgemeinen als Grenze für eine spärliche Kodierung betrachten.
Das grundlegende Problem stellt sich allerdings wie folgt dar: Für nicht sehr umfangreiche Texte degeneriert das Verwenden einer deutlich grösseren Matrix leicht zu einer im klassischen Sinne weniger ausgenutzten assoziativen Struktur, welche allerdings nach Tabelle 3.9 auch mit weniger Auslesefehlern funktionieren kann. Die (extern) komprimierte Darstellung [(40)] nach [Hagström 96] Kap. 3.1.5 kann verwendet werden, um den Speicheraufwand wieder zu reduzieren.
Während dies in einer Simulation oder mit aufwendiger konstruierten Hardware-Elementen eine Option sein kann, stellt dies ein Problem für zwecks optimaler Skalierbarkeit in den Speichermodulen *möglichst „direkt“* [(41)] aufgebaute *Assoziativmaschinen* dar. Auch die Speicherung einer Schlüsselfolge unterliegt dem „Problem vieler falscher Vektoren“, wie die Werte in Tabelle 3.8 nahelegen.
Durch einen dem Auslesen nachfolgenden Korrektur-Schritt durch Analyse des Musters in einem autoassoziativen Memory kann hier eine Verbesserung erzielt werden. Es bietet sich jedoch auch an, zunächst die Idee gleichartiger Kodierung der F und A zu verfolgen, konkret durch innertextliche Sequenzbildung.

3.5. Innertextliche Sequenzbildungen

In [Sommer, Palm 99] (5.2) „Processing of ambiguous initial patterns“

> [...] it is possible to process initial patterns containing an OR-superposition of several parts of stored patterns [...] and successively,

(40) [Palm et al. 10] bezeichnet dies als „synaptic pruning“

(41) [Dierks 05] Kap. 4.2 „Matrixaufbau“: „[...] so erhält man eine ungleich einfachere Lösung für ein neuronales Netz in Matrixstruktur [...] Aus Abb. 18 wird ersichtlich, wie dafür bei der VidAs-Maschine RS-Flip-Flops als bistabile Kippstufen eingesetzt werden“ – in Erweiterung des dort dargestellen Zählerkonzeptes schlage ich allerdings noch den Einsatz einer Kaskade von Addierern für die Berechnung des Schwellvektors mit maximaler Geschwindigkeit $O(ld\,n)$ vor

> a list of retrieved items ordered with respect to the relevance for the given initial pattern can be obtained.

ist zwar erläutert, wie eine Sequenz im Output erhalten werden kann; die beschriebene Methode dient allerdings wieder dem Retrieval, hier der Anreicherung der Ergebnismenge, und nicht der eigentlichen Sequenzerzeugung. Wir werden gleich darauf zurückkommen.

Zurück zu direkten Sequenzen: Jede Zeile ist der Schlüssel für die jeweils nachfolgende Zeile
Assoziationen einer Textfolge

Wir verwenden nun erstmalig eine Sequenz-Kodierung der Text-Daten, d.h. als (F^t) wird keine systematische (deterministisch pseudo-zufällige) Auswahl mit k Indizes verwendet, sondern die jeweils vorhergehende Textzeile [(42)] , d.h.:

$$(\tau_t) = (F^t = T^t, A^t = T^{t+1})$$

Dies ändert zunächst einmal gar nichts an der grundsätzlichen Problematik der fehlerhaft ausgelesenen Antwortvektoren. Eine Korrektur durch eine weitere autoassoziative Matrix kann das Problem lindern, aber nicht vollständig beseitigen.

Der zur Fehlerkorrektur der „falschen Einsen“ notwendige Aufwand ist allerdings bekannt und kann offenbar maximal so gross sein, wie der gesamte zur Textspeicherung notwendige Informationsgehalt.

Eine sich anbietende Strategie zur fehlerlosen Datenspeicherung kann daher sein, diese Korrekturinformation schlicht erneut und – wegen des eben Gesagten – in einer gewiss nicht grösseren, sondern vermutlich kleineren Matrix zu speichern. Dies lässt sich fortsetzen, bis in einer hoffentlich endlichen Anzahl von Speichern schliesslich der gesamte Text reproduzierbar repräsentiert ist.

Allerdings geht hier die Direktheit der bisher verwendeten Kodierung zunächst verloren – es sei denn, man speichert die Fehlerinformation wieder in „lesbarer“ Form. Die kompakteste Angabe zur Fehlerkorrektur ist ja der Index der Teilmenge in der geordneten Aufzählung, welche die Auswahl der korrekten Einsen darstellt. Es ist davon auszugehen, dass bei der Übersetzung dieser Zahl in eine Zeichenkette von Ziffern der Speicheraufwand steigt.

Schlimmer noch lässt sich die tatsächliche Berechnung von (w) wie auf S. 33 am ehesten mit einem herkömmlichen Rechenwerk und Algorithmus vorstellen.

Wir folgen hier also einer anderen Idee.

In [Sommer, Palm 99] heisst es weiter zum *crosswise bidirectional (CB) retrieval* prägnant formuliert auf den Ausleseprozess bezogen:

> The predominating part will be singled out first. Its active components have to be deleted [...] to retrieve the next part. Thus, a segmentation of [...] components is achieved. [...] if parts of several memory patterns with the same size are present [...] [t]his can be achieved by a random deletion processs which singles out one predominate component [...]

(42) $F^0 = T^0$ wird als extern („anderswo“) gespeichert betrachtet.

Wir dagegen nehmen dies sinngemäss bei der Einspeicherung des Textes vorweg. Dort ersetzen wir Komponenten des Musters durch andere, anstatt sie zu löschen, um Mehrdeutigkeiten zu vermeiden.

3.5.1. Vermeidung von Schleifen und allgemeine Transformation

Direkte Sequenzbildung auf kleinem Alphabet, $p = \frac{1}{6}$, $|\{\tau_t\}| = 6$

Die folgenden $\tau_t = (F_t, A_t)$ werden korrekt als $R_t = A[F_t]$ wieder ausgelesen. Kodierung ist wieder „Zeichen auf Position“. Das Alphabet besteht aus 5 Zeichen und alle 6 sechbuchstabigen englischen Worte werden als geschlossener Zyklus von ebensovielen Assoziationen erfolgreich gelernt.

F^t	A^t	R^t
allele	enable	enable
enable	banana	banana
banana	babble	babble
babble	anneal	anneal
anneal	baleen	baleen
baleen	allele	allele

Grenzen direkter Sequenzbildung

Wenn jetzt allerdings ein Wort doppelt erscheint (*Banane* sowohl als Zeile 2 und Zeile 3), funktioniert der Ablauf nicht mehr:

F^t	A^t	R^t
allele	enable	enable
enable	banana	banana
banana	banana	------
banana	babble	------
babble	anneal	anneal

(Dass anschliessend $R_4 = A_4$ wieder korrekt ist, wird hier zwar geprüft und so dargestellt, aber diese Assoziation τ_4 kann natürlich bei einem tatsächlichen Auslesevorgang nicht gefunden werden, solange τ_3 nicht das richtige Ergebnis F_4 ergibt.)

Also zerlegen wir die <**banana**> <**banana**> einmal in

<**ban**> <**ana**> <**banana**> ;

da wir jedoch ein Zeichen aus dem Alphabet für jede Position benötigen (dies ist später beim Auslesen die einzige Möglichkeit, sicher lokal einen Fehler zu erkennen), füllen wir die Zeilen noch auf:

<**banaaa**> <**anaaaa**> <**banana**>

und erhalten dennoch wieder kein korrektes Ergebnis [43] :

F^t	A^t	R^t
allele	enable	enable
enable	banaaa	banaaa
banaaa	anaaaa	------
anaaaa	banana	banana
banana	babble	babble
babble	anneal	anneal

Ist <**aaa**> dem <**ana**> zu ähnlich? (siehe F_2 und F_4) Nein, ...

Alternativ versuchen wir:

<**banaaa**> <**anabbb**> <**banana**> .

F^t	A^t	R^t
allele	enable	enable
enable	banaaa	banaaa
banaaa	anabbb	anabbb
anabbb	banana	banana
banana	babble	babble
babble	anneal	anneal

... denn <**banaaa**> vs. <**banana**> ist *nicht* das Problem! (wieder F_2 und F_4)

Damit haben wir erstmalig einen Text mit ursprünglich vorhandenen Wiederholungen korrekt eingespeichert und auch wieder auslesen können.

D1 *Definitionen zu „Lernen mit* **reblocking**“

Wenn wir die (konstante) Zeilenlänge als `len` bezeichnen, teilen wir dazu, wenn notwendig, eine Zeile in der Mitte oder direkt links davon, so dass linke und rechte Hälften immer diese Längen (im Beispiel (3,3)) haben: [44]

D1

```
. # wie zeile zu teilen ist
. ( len1, len2 ) = ( len//2, len - len//2 )
```

Zu beachten sind dann also die folgenden Regeln [45] :

[43] Dieses Problem fand auch Dierks in [Dierks 05] S.71: „Wählt man als Abfolgewerte für Schleifenvariable welche, die einen zu geringen Ähnlichkeitsabstand zueinander haben, so kann das [...] dazu führen, dass die Werte nicht mehr korrekt memoriert werden.“

[44] Die Zeilenlänge (Blockgrösse) muss hier natürlich `len`≥ 2 sein

[45] Programmbeispiele hier und später in Python, es gilt daher:

- `//` ist die ganzzahlige Division unter Vernachlässigung eines Restes;
- `[a:b]` denotiert bei einer Liste der Länge `len` die Auswahl einer Subsequenz von Stelle `a` (*inclusive*) bis vor Stelle `b` (also *exclusive*), wobei `a=0` bzw. `b=len` gilt, falls nicht angegeben oder ausserhalb des Bereichs gültiger Indexe; negative Werte zählen vom Ende her.
- `+` Listenverkettung, String-Konkatenation

P1 *Algorithmus „Lernen mit* **reblocking***“* [46]

1. Testlernphase: Lerne den Text `the_text` „normal“ in `memory`
2. `memory` prüfen und eventuell `the_text` Modifizieren:

P1
```
. num_errors,check_list = memory.check() # (n,[01]{len})
. if num_errors>0:
.     the_text = flatten(
.         line if correct else # korrekte direkt uebernehmen,
.           # sonst: zwei aufgefuellte teile einfuegen
```
im Beispiel [<**ban**> + <**aaa**> , <**ana**> + <**bbb**>]
```
.         [ line[:len1] + 'a'*len2, line[len1:] + 'b'*len1 ]
.         for line,correct in zip(the_text, check_list)
.     ) # text ausbreiten, so dass eingefuegte paare
.       # normale zeilen werden
```

Das Auslesen geschieht dann durch:

P2 *Algorithmus „Auslesen bei* **reblocking***“* [47]

1. Beginnend mit `line`=*Startschlüssel*, solange Endemarke nicht erreicht:

P2
```
. line = memory[line]
. if len(set(line[-3:]))==1:# wenn letzte drei zeichen gleich:
.     line2 = memory[line]  # teil mit zweiter haelfte holen
.     line = line[:len1] + line2[:len2]
. print line
```

[46] Details zum Lernen:

1. <**a**> und <**b**> sind hier beliebig aber unterschiedlich gewählt, wobei hier sinnvollerweise die Randbedingung zu beachten ist, dass eine Wahl getroffen wird, die nicht mit dem letzten Buchstaben des tatsächlichen Textes übereinstimmt,
2. und weiterhin wird später beim Auslesen ja erforderlich sein, Blöcke wieder zusammenzufügen: deswegen sollen die hinten angefügten Zeichen mindestens `len2`≥ 3 Stück sein; es wird also erst einmal davon ausgegangen, dass derartige Kombinationen im normalen Text nicht vorkommen, und die minimale Zeilenlänge ist `len`≥ 6.

[47] Details zum Auslesen:

1. Der Startschlüssel muss in diesem Modell noch separat gespeichert werden
2. Wir wollen auch einmal annehmen, dass <**anneal**> ein bekanntes Ende des Textes darstellt; alternativ könnte man natürlich bei einem geeigneten Alphabet auch auf beispielsweise das Vorhandensein eines <**.**> überprüfen .

3.5.2. Speicherung eines realen Textes

Im folgenden wird mit verschiedenen Parameterkombinationen geprüft, wieviele Zeilenfolgen gegebener Länge sich in eine Matrix einspeichern lassen, wenn man nun einen *literarischen Text* als Datenmenge betrachtet, welcher bei gegebener *Zeilenlänge c der Zerlegung in einzelne Datensätze* – nicht unbedingt in Übereinstimmung mit den originalen „Textzeilen“ – dann entsprechend T_c Stück Datenzeilen (oder „Worte“, Vektoren) ergibt, zwischen denen die Assoziationen τ_t stattfinden sollen:

Zeilenzerlegungsbeispiel Wir zeigen hier einen Auszug der entsprechend zerlegten Zeilen eines Textes: [48]

F^t	A^t
"\xef\xbb\xbfAlice's A"	'benteuer\r\n\r\n'
'benteuer\r\n\r\n'	'im Wunderlan'
'im Wunderlan'	'd\r\n\r\nvon\r\n\r\n'
'd\r\n\r\nvon\r\n\r\n'	'Lewis Carrol'
'Lewis Carrol'	'l.\r\n\r\nAus de'
'l.\r\n\r\nAus de'	'm Englischen'
'm Englischen'	' von Antonie'
' von Antonie'	' Zimmermann.'
' Zimmermann.'	'\r\n\r\n\r\n\r\n\r\nMi'
'\r\n\r\n\r\n\r\n\r\nMi'	't zweiundvie'
't zweiundvie'	'rzig Illustr'
'rzig Illustr'	'ationen\r\n\r\nv'
'ationen\r\n\r\nv'	'on\r\n\r\nJohn T'
'on\r\n\r\nJohn T'	'enniel.\r\n\r\nA'
'enniel.\r\n\r\nA'	'utorisirte A'
'utorisirte A'	'usgabe.\r\n\r\n\r'
'usgabe.\r\n\r\n\r'	'\nLeipzig\r\n\r\n'
'\nLeipzig\r\n\r\n'	'Johann Fried'
'Johann Fried'	'rich Hartkno'
'rich Hartkno'	'ch.\r\n\r\nOrigi'
'ch.\r\n\r\nOrigi'	'nally publis'
'nally publis'	'hed in 1869.'
'hed in 1869.'	'\r\n\r\n\r\n\r\n\r\n '
'\r\n\r\n\r\n\r\n\r\n '	' O sch\xf6ner,'
' O sch\xf6ner,'	' goldner Nac'
' goldner Nac'	'hmittag,\r\n '
'hmittag,\r\n '	' Wo Flut un'
' Wo Flut un'	'd Himmel lac'
'd Himmel lac'	'ht!\r\n Von'

Die Zeilenlänge ist $c = 12$, „besondere Zeichen“ sind in einer üblichen „\“-Darstellung gezeigt.

[48] vgl. S.91

Beschreibung des Prüfvorgangs

Statt vollständiger Prüfung zur Auffindung des maximal speicherbaren Textumfanges...

...Bisektionsstrategie bezüglich der Anzahl der Datenzeilen

Um den Punkt $T_{\max}$ zu finden, bis zu dem gerade noch ein fehlerfreies Wiederauslesen im Sinne von (v) möglich ist, wären eigentlich zunächst $\{\tau_0\}$, dann $\{\tau_0, \tau_1\} = \{\tau_t | t \leq 1\}$, dann $\{\tau_t | t \leq 2\}$ undsoweiter, also alle Mengen $\{\tau_t | t \leq i\}[i = 0, \ldots, T_c - 1]$, zu lernen und jeweils (v) zu prüfen.
Das maximal mögliche i ist dann $T_{\max} - 1$.
Für grössere T wäre dies natürlich sehr aufwendig; es bietet sich daher an, die Sequenz der i durch eine Bisektionsstrategie zur Überprüfung auszuwählen.
Anfangs wird dabei über eine exponentiell wachsende Folge – empirisch hat sich $a_0 = 1\,;\ a_{i+1} = a_i \cdot 5.6 + 5$ als gut geeignet erwiesen – ein erstes T_{err} gesucht, wo die Korrektheit des Auslesens (v) nicht mehr gegeben ist – im ersten Beispiel 55.
Dort wird dann das Ergebnis $T_{\max} = 43$ nicht nach 43, sondern schon nach 9 Versuchen gefunden, wie das Ablaufprotokoll zeigt.
Es muss dabei jedoch angemerkt werden, dass hier durchaus ein in dem Sinne der obigen Definition falscher Wert gefunden werden *könnte*, denn es ist möglich, dass (v) bei einem (oder mehreren) $i = T_{\max} - 1 - k\,[k > 0]$ nicht gilt.
Die Fehleranzahlen e_i des Matrixspeichers mit (v_i) sind nur bei unkorrelierten Eingangsdaten gemittelt monoton, denn eventuelle „Fehler“ können *theoretisch* durch zusätzlich eingespeicherte Daten „korrigiert“ werden [(49)] . Korrekterweise wäre hier also vielleicht $T'_{\max}$ zu schreiben, worauf wir jedoch im folgenden verzichten.
Jedenfalls jedoch wird auf diesem Wege ein i gefunden werden, welches die Bedingungen der Textspeicherung wie angegeben für einen konkreten Text(-ausschnitt) erfüllt.

[(49)] in allen überprüften Abläufen trat dieses Phänomen jedoch nicht auf

Durchführung

Als Textmenge für dieses erste Beispiel wählen wir

Alice's Abenteuer im Wunderland von *Lewis Carroll* [50]

zunächst in direkter Speicherung [51] , soweit möglich, also ohne die zuvor beschriebene Korrektur.

Zerlegung in V^i erfolgt wie im Beispiel angegeben.

Prüfung der Speicher-Validität (v) Das Ablaufprotokoll (Tab. 3.11) zeigt die überprüften Datenmengen an: Die ersten drei Zeilen entsprechen $a_0 = 1; a_1 = 10; a_2 = \ldots$ wie oben definiert. Danach findet eine binäre Suche des kritischen Wertes zwischen den Grenzen $[10, 55]$ statt, welche hier noch 6 Prüfungen benötigt.

Parameter	Wert
b	89
n	1068
c	12

c	$\lvert(\tau_t)\rvert$	L	e	$(v) = [e = 0]$	$\#_{(a_{ij})}$	$p_{(a_{ij})}$	$\#_{\bigvee(a_{i\cdot})}$	$p_{\bigvee(a_{i\cdot})}$
12	1	0	0	$True$	-1	-0.0	-1	-0.0009
12	10	0	0	$True$	-1	-0.0	-1	-0.0009
12	55	0	1	$False$	-1	-0.0	-1	-0.0009
12	33	0	0	$True$	-1	-0.0	-1	-0.0009
12	44	0	1	$False$	-1	-0.0	-1	-0.0009
12	39	0	0	$True$	-1	-0.0	-1	-0.0009
12	42	0	0	$True$	-1	-0.0	-1	-0.0009
12	43	0	0	$True$	-1	-0.0	-1	-0.0009
12	44	0	1	$False$	-1	-0.0	-1	-0.0009

Tabelle 3.11. *Ablaufprotokoll zur Suche von* $T_{\max}$

Gefundene Maximalanzahl $|t|$ der Assoziationen Gespeichert werden können also bis zu $T_{\max} - 1 = 43$ Assoziationen, das heisst korrekt gespeichert sind $43 \cdot 12 = 516$ Zeichen in den durch den Matrixspeicher gelieferten Antwortvektoren.
Rechts in dieser sowie folgenden Tabellen sind die resultierenden gezählten tatsächlich gesetzten Bits und zugehörigen Bitdichten im Matrixspeicher insgesamt angegeben; ausserdem diese Werte für die Nutzung einer Matrixzeile – also der Anteil der Zeilen, welche mindestens durch einen der Vektoren aktiviert wurde.

Wahl der Zeilenlänge Wie in der Parameterliste angegeben, ist die Zeilenlänge $c = 12$ dabei zunächst willkürlich gewählt. b ergibt sich aus dem im (gesamten) Text verwendeten Alphabet [52] , und damit auch n. Im nächsten

[50] [Carroll 69] Aus dem Englischen von Antonie Zimmermann; vgl. S. 91
[51] also noch ohne l-Transformation durch $P1$,$P2$, daher immer $L = 0$ Korrekturebenen
[52] Anzahl verschiedener *Unicode-Codepoints*

Versuch (Tab. 3.12) wählen wir nun zum Vergleich das minimale [53] $c = 6$ und erhalten entsprechend ein schlechteres Ergebnis:

Parameter	Wert
b	89
n	534
c	6

c	$\lvert(\tau_t)\rvert$	L	e	$(v) = [e = 0]$	$\#_{(a_{ij})}$	$p_{(a_{ij})}$	$\#_{\bigvee(a_{i\cdot})}$	$p_{\bigvee(a_{i\cdot})}$
6	1	0	0	$True$	-1	-0.0	-1	-0.0019
6	10	0	0	$True$	-1	-0.0	-1	-0.0019
6	55	0	5	$False$	-1	-0.0	-1	-0.0019
6	33	0	2	$False$	-1	-0.0	-1	-0.0019
6	22	0	0	$True$	-1	-0.0	-1	-0.0019
6	28	0	2	$False$	-1	-0.0	-1	-0.0019
6	25	0	0	$True$	-1	-0.0	-1	-0.0019
6	27	0	2	$False$	-1	-0.0	-1	-0.0019
6	26	0	0	$True$	-1	-0.0	-1	-0.0019
6	27	0	2	$False$	-1	-0.0	-1	-0.0019

Tabelle 3.12. *Suche von $T_{\max}$ mit anderem c, n*

Gespeichert werden können hier nur $T_{\max}-1 = 26$ Assoziationen, also $26{\cdot}6 = 156$ Zeichen. [54] .

Korrekturfunktion bis zu 2 Ebenen tief

Das als $P1$ im Abschnitt 3.5.1 definierte Verfahren mit im Fehlerfalle modifizierten Assoziationsfolgen $(\tau_t) \to (\tau'_{t'})$ kann nun angewandt werden, um den Matrixspeicher besser auszunutzen.
Sollten dann immer noch Fehler auftreten, so kann die Lerndatenfolge erneut modifiziert werden: $(\tau'_{t'}) \to (\tau''_{t''})$
Diese hier $L \leq L_{\max} = 2$ *Korrektur-Ebenen* weit reichende Modifikation ist eine Form der auf Seite 19 definierten Translation (l).
Das Protokoll ergibt, dass unter Anwendung dieses Translations-Algorithmus nun auf die τ_t bezogen 112 Assoziationen eingespeichert werden können:

Parameter	Wert
b	89
n	534
c	6
$L_{\max}$	2

c	$\lvert\{\tau^{(L)}_{t^{(L)}}\}\rvert$	L	e	$(v) = [e = 0]$	$\#_{(a_{ij})}$	$p_{(a_{ij})}$	$\#_{\bigvee(a_{i\cdot})}$	$p_{\bigvee(a_{i\cdot})}$
6	1	0	0	$True$	-1	-0.0	-1	-0.0019
6	10	0	0	$True$	-1	-0.0	-1	-0.0019
6	55	0	5	$False$	-1	-0.0	-1	-0.0019
6	59	1	1	$False$	-1	-0.0	-1	-0.0019
6	59	2	0	$True$	-1	-0.0	-1	-0.0019
6	249	0	128	$False$	-1	-0.0	-1	-0.0019
6	376	1	172	$False$	-1	-0.0	-1	-0.0019
6	547	2	310	$False$	-1	-0.0	-1	-0.0019

[53] vgl. “Details zum Lernen“ auf Seite 57

[54] Zu sehen ist hier auch im Gegensatz zum vorigen Experiment, dass bereits um $i = 50$ herum deutlich mehr Fehler auftreten als nur *einer*.

6	168	0	58	*False*	−1	−0.0	−1	−0.0019
6	225	1	45	*False*	−1	−0.0	−1	−0.0019
6	269	2	53	*False*	−1	−0.0	−1	−0.0019
6	112	0	21	*False*	−1	−0.0	−1	−0.0019
6	132	1	5	*False*	−1	−0.0	−1	−0.0019
6	136	2	0	*True*	−1	−0.0	−1	−0.0019
6	140	0	41	*False*	−1	−0.0	−1	−0.0019
6	180	1	23	*False*	−1	−0.0	−1	−0.0019
6	202	2	20	*False*	−1	−0.0	−1	−0.0019
6	126	0	34	*False*	−1	−0.0	−1	−0.0019
6	159	1	17	*False*	−1	−0.0	−1	−0.0019
6	175	2	16	*False*	−1	−0.0	−1	−0.0019
6	119	0	29	*False*	−1	−0.0	−1	−0.0019
6	147	1	10	*False*	−1	−0.0	−1	−0.0019
6	156	2	5	*False*	−1	−0.0	−1	−0.0019
6	116	0	24	*False*	−1	−0.0	−1	−0.0019
6	139	1	6	*False*	−1	−0.0	−1	−0.0019
6	144	2	5	*False*	−1	−0.0	−1	−0.0019
6	114	0	22	*False*	−1	−0.0	−1	−0.0019
6	135	1	10	*False*	−1	−0.0	−1	−0.0019
6	144	2	5	*False*	−1	−0.0	−1	−0.0019
6	113	0	22	*False*	−1	−0.0	−1	−0.0019
6	134	1	5	*False*	−1	−0.0	−1	−0.0019
6	138	2	3	*False*	−1	−0.0	−1	−0.0019

Es bietet sich eine andere geordnete Darstellung des Ablaufes an (Tab. 3.13):

$\lvert(\tau_t)\rvert$	$e\,(L=0)$	$e\,(L=1)$	$e\,(L=2)$
1	0		
10	0		
55	5	1 ($\lvert\left(\tau'_{t'}\right)\rvert = 59$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 59$)
112	21	5 ($\lvert\left(\tau'_{t'}\right)\rvert = 132$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 136$)
113	22	5 ($\lvert\left(\tau'_{t'}\right)\rvert = 134$)	3 ($\lvert\left(\tau''_{t''}\right)\rvert = 138$)
114	22	10 ($\lvert\left(\tau'_{t'}\right)\rvert = 135$)	5 ($\lvert\left(\tau''_{t''}\right)\rvert = 144$)
116	24	6 ($\lvert\left(\tau'_{t'}\right)\rvert = 139$)	5 ($\lvert\left(\tau''_{t''}\right)\rvert = 144$)
119	29	10 ($\lvert\left(\tau'_{t'}\right)\rvert = 147$)	5 ($\lvert\left(\tau''_{t''}\right)\rvert = 156$)
126	34	17 ($\lvert\left(\tau'_{t'}\right)\rvert = 159$)	16 ($\lvert\left(\tau''_{t''}\right)\rvert = 175$)
140	41	23 ($\lvert\left(\tau'_{t'}\right)\rvert = 180$)	20 ($\lvert\left(\tau''_{t''}\right)\rvert = 202$)
168	58	45 ($\lvert\left(\tau'_{t'}\right)\rvert = 225$)	53 ($\lvert\left(\tau''_{t''}\right)\rvert = 269$)
280	128	172 ($\lvert\left(\tau'_{t'}\right)\rvert = 376$)	310 ($\lvert\left(\tau''_{t''}\right)\rvert = 547$)

Tabelle 3.13. *Fehlerhäufigkeiten im Beispiel nach Anzahl Assoziationen und Korrekturtiefe*

Nicht vorhandene Werte bedeuten, dass bereits auf der vorigen Ebene kein Fehler mehr auftrat; $e > 0$ in der ganz rechten Spalte dagegen, dass auch mit maximaler Korrektur kein korrektes Auslesen mehr möglich war.
Die Verbesserung gegenüber dem Ergebnis 26 von vorher jetzt auf 112 ist deutlich.
Zum Vergleich noch dasselbe mit $c = 12$ nur in abgekürzter Form (Tab. 3.14); 137 Assoziationen können gespeichert werden:

Parameter	Wert
b	89
n	1068
c	12
$L_{\max}$	2

$\lvert(\tau_t)\rvert$	$e\,(L=0)$	$e\,(L=1)$	$e\,(L=2)$
1	0		
10	0		
55	1	0 ($\lvert\left(\tau'_{t'}\right)\rvert = 55$)	
112	3	1 ($\lvert\left(\tau'_{t'}\right)\rvert = 114$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 114$)
126	5	3 ($\lvert\left(\tau'_{t'}\right)\rvert = 130$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 132$)
133	7	0 ($\lvert\left(\tau'_{t'}\right)\rvert = 139$)	
137	7	1 ($\lvert\left(\tau'_{t'}\right)\rvert = 143$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 143$)
138	7	4 ($\lvert\left(\tau'_{t'}\right)\rvert = 144$)	1 ($\lvert\left(\tau''_{t''}\right)\rvert = 147$)
139	7	5 ($\lvert\left(\tau'_{t'}\right)\rvert = 145$)	1 ($\lvert\left(\tau''_{t''}\right)\rvert = 149$)
140	7	3 ($\lvert\left(\tau'_{t'}\right)\rvert = 146$)	1 ($\lvert\left(\tau''_{t''}\right)\rvert = 148$)
168	17	5 ($\lvert\left(\tau'_{t'}\right)\rvert = 184$)	3 ($\lvert\left(\tau''_{t''}\right)\rvert = 188$)
280	83	111 ($\lvert\left(\tau'_{t'}\right)\rvert = 362$)	195 ($\lvert\left(\tau''_{t''}\right)\rvert = 472$)

Tabelle 3.14. *Fehlerhäufigkeiten im Beispiel f. $c = 12$*

Im $P1$-Algorithmus wird im Beispiel mit jeweils 3× demselben Füllzeichen eine Ergänzung der beiden Halbzeilen vorgenommen.
Unter der Annahme, dass keine 3 gleichen Zeichen [(55)] im Originaltext vorkommen, ist diese erste Translation

$$l_1 = [L = 0 \rightarrow L = 1]$$

[(55)] mehr dazu in Abschnitt 3.5.4

eine Bijektion.

Ist l_2 eine Bijektion?

In der Folge $\left(\tau'_{t'}\right)$ ist diese Bedingung dann jedoch offenbar nicht mehr gegeben. Im allgemeinen wird man nicht mehr unbedingt beim Auslesen des Speichers angeben können, durch welches l_i eine bestimmte Ersetzung vorgenommen wurde. Während l_2 durchaus eine Bijektion *ist*, kann die Rück-Transformation

$$l_2^{-1} = [L = 2 \to L = 1]$$

also mit den im Matrixspeicher zur Verfügung stehenden Daten nicht mehr einzeln durchgeführt werden.
Wie man sich jedoch einfach überlegen kann, *ist*

$$l' = l_{L_{\max}} \circ \ldots \circ l_1$$

bijektiv und die wiederholte Anwendung von $P2$ liefert

$$l'^{-1} = [L = L_{\max} \to L = 0]$$

Varianten beim Füllen

und damit das gewünschte Original in dekodierbarer Form.
Die Uniformität der Füllzeichen ist bei $c = 6$ so erforderlich, wenn man zumindest einfache Zeichenwiederholungen im Originaltext berücksichtigen können möchte.

3.5.3. Variation

Im letzten Beispiel mit $c = 12$ kann man diese strenge Bedingung etwas lockern, indem man die jetzt $\frac{c}{2} = 6$ Füllzeichen jetzt aus $v_{pad} = 2$ Zeichen alternierend bestehen lässt; man erkennt dann beim Auslesen die vorzunehmende Rücktransformation durch die Modifikation

P2' *P2'* *Algorithmus „Auslesen bei* **lockerem reblocking**“

```
. if len(set(line[-6:]))==2:# Menge letzter 6 Z. hat Umfang 2
```

und erhält im Versuch, dass $|(\tau_t)| = 182$ Assoziationen gespeichert werden können:

Parameter	Wert
b	89
n	1068
c	12
v_{pad}	2
$L_{\max}$	2

$\lvert(\tau_t)\rvert$	$e\,(L=0)$	$e\,(L=1)$	$e\,(L=2)$
1	0		
10	0		
55	1	0 ($\lvert\left(\tau'_{t'}\right)\rvert = 55$)	
168	17	4 ($\lvert\left(\tau'_{t'}\right)\rvert = 184$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 187$)
182	24	4 ($\lvert\left(\tau'_{t'}\right)\rvert = 205$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 208$)
183	24	8 ($\lvert\left(\tau'_{t'}\right)\rvert = 206$)	3 ($\lvert\left(\tau''_{t''}\right)\rvert = 213$)
184	24	5 ($\lvert\left(\tau'_{t'}\right)\rvert = 207$)	1 ($\lvert\left(\tau''_{t''}\right)\rvert = 211$)
186	24	6 ($\lvert\left(\tau'_{t'}\right)\rvert = 209$)	2 ($\lvert\left(\tau''_{t''}\right)\rvert = 214$)
189	25	12 ($\lvert\left(\tau'_{t'}\right)\rvert = 213$)	2 ($\lvert\left(\tau''_{t''}\right)\rvert = 224$)
196	30	11 ($\lvert\left(\tau'_{t'}\right)\rvert = 225$)	4 ($\lvert\left(\tau''_{t''}\right)\rvert = 235$)
224	46	25 ($\lvert\left(\tau'_{t'}\right)\rvert = 269$)	17 ($\lvert\left(\tau''_{t''}\right)\rvert = 293$)
280	83	48 ($\lvert\left(\tau'_{t'}\right)\rvert = 362$)	54 ($\lvert\left(\tau''_{t''}\right)\rvert = 409$)

Die vollständige Tabelle der Ersetzungen durch l_1 ist [56] [57] : l_1:

e	t	$F^{(t)}$	$[F'^{(t+e)}, F'^{(t+1+e)}]$
0	12	'ationen\r\n\r\nv'	['apapapatione', 'n\r\n\r\nvjpjpjp']
1	16	'usgabe.\r\n\r\n\r'	['SOSOSOusgabe', '.\r\n\r\n\r.f.f.f']
2	23	'\r\n\r\n\r\n\r\n\r\n\r\n '	['dododo\r\n\r\n\r\n', '\r\n\r\n dRdRdR']
3	34	'tschern sach'	['&d&d&dtscher', 'n sachg-g-g-']
4	53	' Sie triebe'	['7O7O7O Sie ', 'triebe,\xfc,\xfc,\xfc']
5	55	'r\xe4ngten ihn,'	["'i'i'ir\xe4ngte", 'n ihn,x&x&x&']
6	57	'hrchen zu er'	['PMPMPMhrchen', ' zu er\xf6\xef\xf6\xef\xf6\xef']
7	59	' Die erste'	['[m[m[m Die', ' erstex*x*x*']
8	77	'hen sie vom '	['*c*c*chen si', 'e vom \xabj\xabj\xabj']
9	88	'len sich so '	['\xbb1\xbb1\xbb1len si', 'ch so ;3;3;3']
10	100	'rige ein and'	['V\xefV\xefV\xefrige e', 'in and4M4M4M']
11	101	'er Mal!\xab\r\n '	['OVOVOVer Mal', '!\xab\r\n ?M?M?M']
12	102	' O nein, si'	['\nT\nT\nT O ne', 'in, si*a*a*a']
13	106	'chon ein and'	['JLJLJLchon e', 'in and?w?w?w']
14	113	'erland\r\n '	['DwDwDwerland', '\r\n &c&c&c']
15	127	'cht.\r\n\r\n '	['3\xbf3\xbf3\xbfcht.\r\n', '\r\n !u!u!u']
16	149	'ng gegen die'	['oIoIoIng geg', 'en die&i&i&i']
17	150	' Uebersetzer'	['kqkqkq Ueber', 'setzergtgtgt']
18	152	'echen, die e'	['IdIdIdechen,', ' die e\xab=\xab=\xab=']
19	153	'inige einges'	['7?7?7?inige ', 'einges\ru\ru\ru']
20	158	' der deutsch'	['2\n2\n2\n der d', 'eutsch:O:O:O']
21	166	'edichten ers'	['\xabE\xabE\xabEedicht', 'en ersw\xbfw\xbfw\xbf']
22	179	'ndtheit der\r'	['\xf9]\xf9]\xf9]ndthei', 't der\reWeWeW']
23	180	'\nUebersetzer'	['qhqhqh\nUeber', 'setzer*J*J*J']

Anzumerken ist zu all diesen Experimenten, dass die zufällige Wahl der Füllzeichen so im Einzelfalle möglicherweise ungünstig, jedenfalls aber nur beispielhaft ist:
Wie die Abfolge der $e(L = 1) = [0, 0, 1, 5, 5, 10, 6, 10, \ldots]$ in Tabelle 3.13 zeigt, ist bei diesem sehr einfachen Korrekturprinzip die Leistung offenbar noch sehr durch die Zufalls-Auswahl bestimmt.
Wir könnten, bevor wir zu Modifikationen des Verfahrens und zu Kombinationen mit deutlich anderen Kodierungen voranschreiten, noch

1. einmal nur das am seltensten vorkommende Zeichen als Füllzeichen wählen (Alphabet-Nutzungsrate $r = 1/b$), oder
2. einen gewissen Anteil des nach Häufigkeit [58] sortierten Alphabetes wiederholt verwenden ($r = p_{\texttt{fill}} \cdot b$).

In beiden Fällen wird hierdurch das Vorgehen auch deterministisch.
Man kann jedoch durch Vergleich mit einer Mittelung über mehrere Läufe mit unterschiedlichen Startwerten für den Zufallsgenerator zuerst abschätzen, ob diese Auswahl der Füllzeichen tatsächlich wesentlich ist. Wir wiederholen daher

[56] $e = 0, \ldots$ zählt dabei die Fehler mit. Die zwischendurch (nicht dargestellten) korrekt auslesbaren Werte $F^{(t)}$ werden auch entsprechend dem aktuellen e mit verschoben auf ein jeweiliges $F'^{(t+e)}$.

[57] bei den linken Hälften der Ersetzungen ist hier abweichend vom beschriebenen Algorithmus der Fülltext links eingefügt

[58] und bei Gleichhäufigkeit alphabetisch geordnet

genau das letzte Experiment mit verschiedenen Startwerten für den Pseudo-Zufallsgenerator und erhalten diese Werte als maximal speicherbare Assoziationsanzahlen $T_{\max}$:

Parameter	Wert
b	89
n	1068
c	12
v_{pad}	2
$L_{\max}$	2

Füllzeichenwahl

162 174 167 167 150 172 161 172 179 172 172 161 168 164 189 182 159

Es ergeben sich der Mittelwert $m_{T_{\max}} = 168.88$ und die Streuung $s_{T_{\max}} = 9.08$. Diese Streuung ist nicht gering, aber auch nicht kritisch: Mit $3\sigma \approx 0.15 \cdot T$ sollte es keine Auswirkungen bei der Bewertung der grundsätzlichen Nutzbarkeit des Verfahrens haben können, sobald später grössere Textmengen eingespeichert werden können. Insofern ist die Auswahl der Füllzeichen anscheinend unkritisch.

Zeilenlängen Mit jeweils vier unterschiedlichen Startwerten des Zufallsgenerators betrachten wir nun den Effekt der unterschiedlichen Zeilenlängen c.
Zu beachten bei der Interpretation des Umfanges der möglichen Assoziationen ist, dass deren Anzahl $|\{\tau_t\}|$ natürlich mit der Zeilenlänge c zu gewichten (multiplizieren) ist, um einen Messwert für die gespeicherte Information zu erhalten.

Parameter	Wert
b	89
n	534
c	6
v_{pad}	2
$L_{\max}$	2

111 98 114 121 ; $m_{T_{\max}} = 111.00$; $s_{T_{\max}} = 8.34$; $m_{T_{\max} \cdot c} = 666.00$

Übersicht der Fehlerzahlen eines der Läufe:

$\lvert(\tau_t)\rvert$	$e\,(L=0)$	$e\,(L=1)$	$e\,(L=2)$
1	0		
10	0		
55	5	0 ($\lvert(\tau'_{t'})\rvert = 59$)	
112	21	0 ($\lvert(\tau'_{t'})\rvert = 132$)	
119	29	2 ($\lvert(\tau'_{t'})\rvert = 147$)	0 ($\lvert(\tau''_{t''})\rvert = 148$)
121	29	5 ($\lvert(\tau'_{t'})\rvert = 149$)	0 ($\lvert(\tau''_{t''})\rvert = 153$)
122	29	6 ($\lvert(\tau'_{t'})\rvert = 150$)	2 ($\lvert(\tau''_{t''})\rvert = 155$)
123	31	6 ($\lvert(\tau'_{t'})\rvert = 153$)	1 ($\lvert(\tau''_{t''})\rvert = 158$)
126	34	6 ($\lvert(\tau'_{t'})\rvert = 159$)	1 ($\lvert(\tau''_{t''})\rvert = 164$)
140	41	10 ($\lvert(\tau'_{t'})\rvert = 180$)	3 ($\lvert(\tau''_{t''})\rvert = 189$)
168	58	26 ($\lvert(\tau'_{t'})\rvert = 225$)	11 ($\lvert(\tau''_{t''})\rvert = 250$)
280	158	132 ($\lvert(\tau'_{t'})\rvert = 437$)	163 ($\lvert(\tau''_{t''})\rvert = 568$)

In dieser Tabelle sehen wir, dass für 121 eingespeicherte Assoziationen (bezogen auf den Originaltext) ein fehlerloses Auslesen möglich ist. Die Zeilen darunter illustrieren die Fehlerrate bei dem Versuch, mehr als dieses mögliche Maximum an Daten einzuschreiben. Bemerkenswert ist, dass (siehe letzte Zeile „280“) mit weiteren Korrekturebenen die Anzahl der Fehler wieder zunimmt, weshalb wir unsere Experimente dann auf 2 Ebenen beschränkt haben.
Sukzessive wird im folgenden die Zeilenlänge erhöht. Es ist wieder immer die im Mittel maximal speicherbare Assoziationszahl $m_{T_{\max}}$ und resultierende Zeichenanzahl $m_{T_{\max} \cdot c}$ angegeben.

Die Ablauftabelle ist jeweils abgedruckt für den letzten Durchlauf (im obigen Beispiel mit den Ergebnissen „111, 98, 114, 121“ findet sich deswegen in der Zeile „121“ zum letzten Mal eine 0).

Parameter	Wert
b	89
n	712
c	8
v_{pad}	2
$L_{\max}$	2

112 113 118 141 ; $m_{T_{\max}} = 121.00$; $s_{T_{\max}} = 11.77$; $m_{T_{\max} \cdot c} = 968.00$

Übersicht der Fehlerzahlen eines der Läufe:

$\lvert(\tau_t)\rvert$	$e\,(L=0)$	$e\,(L=1)$	$e\,(L=2)$
1	0		
10	0		
55	0		
112	7	2 ($\lvert\left(\tau'_{t'}\right)\rvert = 118$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 119$)
140	21	4 ($\lvert\left(\tau'_{t'}\right)\rvert = 160$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 163$)
141	21	5 ($\lvert\left(\tau'_{t'}\right)\rvert = 161$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 165$)
142	21	8 ($\lvert\left(\tau'_{t'}\right)\rvert = 162$)	4 ($\lvert\left(\tau''_{t''}\right)\rvert = 169$)
144	21	6 ($\lvert\left(\tau'_{t'}\right)\rvert = 164$)	2 ($\lvert\left(\tau''_{t''}\right)\rvert = 169$)
147	21	9 ($\lvert\left(\tau'_{t'}\right)\rvert = 167$)	3 ($\lvert\left(\tau''_{t''}\right)\rvert = 175$)
154	28	10 ($\lvert\left(\tau'_{t'}\right)\rvert = 181$)	6 ($\lvert\left(\tau''_{t''}\right)\rvert = 190$)
168	40	19 ($\lvert\left(\tau'_{t'}\right)\rvert = 207$)	9 ($\lvert\left(\tau''_{t''}\right)\rvert = 225$)
280	118	82 ($\lvert\left(\tau'_{t'}\right)\rvert = 397$)	104 ($\lvert\left(\tau''_{t''}\right)\rvert = 478$)

Parameter	Wert
b	89
n	890
c	10
v_{pad}	2
$L_{\max}$	2

130 148 149 132 ; $m_{T_{\max}} = 139.75$; $s_{T_{\max}} = 8.79$; $m_{T_{\max} \cdot c} = 1397.50$

Übersicht der Fehlerzahlen eines der Läufe:

$\lvert(\tau_t)\rvert$	$e\,(L=0)$	$e\,(L=1)$	$e\,(L=2)$
1	0		
10	0		
55	2	0 ($\lvert\left(\tau'_{t'}\right)\rvert = 56$)	
112	5	1 ($\lvert\left(\tau'_{t'}\right)\rvert = 116$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 116$)
126	8	1 ($\lvert\left(\tau'_{t'}\right)\rvert = 133$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 133$)
130	9	2 ($\lvert\left(\tau'_{t'}\right)\rvert = 138$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 139$)
132	9	3 ($\lvert\left(\tau'_{t'}\right)\rvert = 140$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 142$)
133	9	2 ($\lvert\left(\tau'_{t'}\right)\rvert = 141$)	1 ($\lvert\left(\tau''_{t''}\right)\rvert = 142$)
140	11	3 ($\lvert\left(\tau'_{t'}\right)\rvert = 150$)	2 ($\lvert\left(\tau''_{t''}\right)\rvert = 152$)
168	18	5 ($\lvert\left(\tau'_{t'}\right)\rvert = 185$)	6 ($\lvert\left(\tau''_{t''}\right)\rvert = 189$)
280	100	78 ($\lvert\left(\tau'_{t'}\right)\rvert = 379$)	72 ($\lvert\left(\tau''_{t''}\right)\rvert = 456$)

Parameter	Wert
b	89
n	1068
c	12
v_{pad}	2
$L_{\max}$	2

175 172 167 159 ; $m_{T_{\max}} = 168.25$; $s_{T_{\max}} = 6.06$; $m_{T_{\max} \cdot c} = 2019.00$

Übersicht der Fehlerzahlen eines der Läufe:

$\lvert(\tau_t)\rvert$	$e(L=0)$	$e(L=1)$	$e(L=2)$
1	0		
10	0		
55	1	0 ($\lvert(\tau'_{t'})\rvert = 55$)	
112	3	1 ($\lvert(\tau'_{t'})\rvert = 114$)	0 ($\lvert(\tau''_{t''})\rvert = 114$)
140	7	0 ($\lvert(\tau'_{t'})\rvert = 146$)	
144	9	3 ($\lvert(\tau'_{t'})\rvert = 152$)	0 ($\lvert(\tau''_{t''})\rvert = 154$)
146	9	3 ($\lvert(\tau'_{t'})\rvert = 154$)	0 ($\lvert(\tau''_{t''})\rvert = 156$)
147	9	2 ($\lvert(\tau'_{t'})\rvert = 155$)	0 ($\lvert(\tau''_{t''})\rvert = 156$)
148	10	2 ($\lvert(\tau'_{t'})\rvert = 157$)	0 ($\lvert(\tau''_{t''})\rvert = 158$)
149	10	1 ($\lvert(\tau'_{t'})\rvert = 158$)	0 ($\lvert(\tau''_{t''})\rvert = 158$)
150	11	3 ($\lvert(\tau'_{t'})\rvert = 160$)	0 ($\lvert(\tau''_{t''})\rvert = 162$)
151	11	1 ($\lvert(\tau'_{t'})\rvert = 161$)	0 ($\lvert(\tau''_{t''})\rvert = 161$)
152	12	2 ($\lvert(\tau'_{t'})\rvert = 163$)	0 ($\lvert(\tau''_{t''})\rvert = 164$)
153	13	1 ($\lvert(\tau'_{t'})\rvert = 165$)	0 ($\lvert(\tau''_{t''})\rvert = 165$)
154	14	5 ($\lvert(\tau'_{t'})\rvert = 167$)	0 ($\lvert(\tau''_{t''})\rvert = 171$)
155	14	2 ($\lvert(\tau'_{t'})\rvert = 168$)	0 ($\lvert(\tau''_{t''})\rvert = 169$)
156	14	3 ($\lvert(\tau'_{t'})\rvert = 169$)	0 ($\lvert(\tau''_{t''})\rvert = 171$)
157	14	2 ($\lvert(\tau'_{t'})\rvert = 170$)	0 ($\lvert(\tau''_{t''})\rvert = 171$)
158	14	3 ($\lvert(\tau'_{t'})\rvert = 171$)	0 ($\lvert(\tau''_{t''})\rvert = 173$)
159	15	3 ($\lvert(\tau'_{t'})\rvert = 173$)	0 ($\lvert(\tau''_{t''})\rvert = 175$)
160	15	5 ($\lvert(\tau'_{t'})\rvert = 174$)	2 ($\lvert(\tau''_{t''})\rvert = 178$)
168	17	4 ($\lvert(\tau'_{t'})\rvert = 184$)	1 ($\lvert(\tau''_{t''})\rvert = 187$)
280	83	61 ($\lvert(\tau'_{t'})\rvert = 362$)	62 ($\lvert(\tau''_{t''})\rvert = 422$)

Parameter	Wert
b	89
n	1246
c	14
v_{pad}	2
$L_{\max}$	2

167 172 166 169 ; $m_{T_{\max}} = 168.50$; $s_{T_{\max}} = 2.29$; $m_{T_{\max} \cdot c} = 2359.00$

Übersicht der Fehlerzahlen eines der Läufe:

$\lvert(\tau_t)\rvert$	$e(L=0)$	$e(L=1)$	$e(L=2)$
1	0		
10	0		
55	0		
112	0		
140	2	0 ($\lvert(\tau'_{t'})\rvert = 141$)	
154	3	0 ($\lvert(\tau'_{t'})\rvert = 156$)	
161	5	1 ($\lvert(\tau'_{t'})\rvert = 165$)	0 ($\lvert(\tau''_{t''})\rvert = 165$)

165	7	0 $(\lvert \left(\tau'_{t'}\right) \rvert = 171)$	
167	7	0 $(\lvert \left(\tau'_{t'}\right) \rvert = 173)$	
168	7	5 $(\lvert \left(\tau'_{t'}\right) \rvert = 174)$	0 $(\lvert \left(\tau''_{t''}\right) \rvert = 178)$
169	8	1 $(\lvert \left(\tau'_{t'}\right) \rvert = 176)$	0 $(\lvert \left(\tau''_{t''}\right) \rvert = 176)$
170	9	3 $(\lvert \left(\tau'_{t'}\right) \rvert = 178)$	1 $(\lvert \left(\tau''_{t''}\right) \rvert = 180)$
280	66	34 $(\lvert \left(\tau'_{t'}\right) \rvert = 345)$	33 $(\lvert \left(\tau''_{t''}\right) \rvert = 378)$

Parameter	Wert
b	89
n	1602
c	18
v_{pad}	2
$L_{\max}$	2

175 181 203 214 ; $m_{T_{\max}} = 193.25$; $s_{T_{\max}} = 15.88$; $m_{T_{\max} \cdot c} = 3478.50$

Übersicht der Fehlerzahlen eines der Läufe:

$\lvert (\tau_t) \rvert$	$e\,(L=0)$	$e\,(L=1)$	$e\,(L=2)$
1	0		
10	0		
55	0		
168	10	1 $(\lvert \left(\tau'_{t'}\right) \rvert = 177)$	0 $(\lvert \left(\tau''_{t''}\right) \rvert = 177)$
196	13	3 $(\lvert \left(\tau'_{t'}\right) \rvert = 208)$	0 $(\lvert \left(\tau''_{t''}\right) \rvert = 210)$
210	15	5 $(\lvert \left(\tau'_{t'}\right) \rvert = 224)$	0 $(\lvert \left(\tau''_{t''}\right) \rvert = 228)$
214	16	2 $(\lvert \left(\tau'_{t'}\right) \rvert = 229)$	0 $(\lvert \left(\tau''_{t''}\right) \rvert = 230)$
215	16	4 $(\lvert \left(\tau'_{t'}\right) \rvert = 230)$	1 $(\lvert \left(\tau''_{t''}\right) \rvert = 233)$
216	16	2 $(\lvert \left(\tau'_{t'}\right) \rvert = 231)$	1 $(\lvert \left(\tau''_{t''}\right) \rvert = 232)$
217	16	5 $(\lvert \left(\tau'_{t'}\right) \rvert = 232)$	4 $(\lvert \left(\tau''_{t''}\right) \rvert = 236)$
224	19	3 $(\lvert \left(\tau'_{t'}\right) \rvert = 242)$	2 $(\lvert \left(\tau''_{t''}\right) \rvert = 244)$
280	45	26 $(\lvert \left(\tau'_{t'}\right) \rvert = 324)$	17 $(\lvert \left(\tau''_{t''}\right) \rvert = 349)$

Parameter	Wert
b	89
n	2136
c	24
v_{pad}	2
$L_{\max}$	2

262 240 257 254 ; $m_{T_{\max}} = 253.25$; $s_{T_{\max}} = 8.17$; $m_{T_{\max} \cdot c} = 6078.00$

Übersicht der Fehlerzahlen eines der Läufe:

$\lvert (\tau_t) \rvert$	$e\,(L=0)$	$e\,(L=1)$	$e\,(L=2)$
1	0		
10	0		
55	0		
168	4	0 $(\lvert \left(\tau'_{t'}\right) \rvert = 171)$	
224	11	2 $(\lvert \left(\tau'_{t'}\right) \rvert = 234)$	0 $(\lvert \left(\tau''_{t''}\right) \rvert = 235)$
252	14	5 $(\lvert \left(\tau'_{t'}\right) \rvert = 265)$	0 $(\lvert \left(\tau''_{t''}\right) \rvert = 269)$
253	14	5 $(\lvert \left(\tau'_{t'}\right) \rvert = 266)$	0 $(\lvert \left(\tau''_{t''}\right) \rvert = 270)$
254	14	4 $(\lvert \left(\tau'_{t'}\right) \rvert = 267)$	0 $(\lvert \left(\tau''_{t''}\right) \rvert = 270)$
255	14	4 $(\lvert \left(\tau'_{t'}\right) \rvert = 268)$	1 $(\lvert \left(\tau''_{t''}\right) \rvert = 271)$
256	14	5 $(\lvert \left(\tau'_{t'}\right) \rvert = 269)$	3 $(\lvert \left(\tau''_{t''}\right) \rvert = 273)$
259	14	3 $(\lvert \left(\tau'_{t'}\right) \rvert = 272)$	1 $(\lvert \left(\tau''_{t''}\right) \rvert = 274)$

266	17	4 ($\lvert \left(\tau'_{t'}\right) \rvert = 282$)	2 ($\lvert \left(\tau''_{t''}\right) \rvert = 285$)
280	19	4 ($\lvert \left(\tau'_{t'}\right) \rvert = 298$)	3 ($\lvert \left(\tau''_{t''}\right) \rvert = 301$)

Parameter	Wert
b	89
n	2670
c	30
v_{pad}	2
$L_{\max}$	2

321 321 324 314 ; $m_{T_{\max}} = 320.00$; $s_{T_{\max}} = 3.67$; $m_{T_{\max} \cdot c} = 9600.00$

Übersicht der Fehlerzahlen eines der Läufe:

$\lvert (\tau_t) \rvert$	$e\,(L=0)$	$e\,(L=1)$	$e\,(L=2)$
1	0		
10	0		
55	0		
280	7	0 ($\lvert \left(\tau'_{t'}\right) \rvert = 286$)	
298	8	2 ($\lvert \left(\tau'_{t'}\right) \rvert = 305$)	0 ($\lvert \left(\tau''_{t''}\right) \rvert = 306$)
307	8	1 ($\lvert \left(\tau'_{t'}\right) \rvert = 314$)	0 ($\lvert \left(\tau''_{t''}\right) \rvert = 314$)
312	8	1 ($\lvert \left(\tau'_{t'}\right) \rvert = 319$)	0 ($\lvert \left(\tau''_{t''}\right) \rvert = 319$)
314	8	3 ($\lvert \left(\tau'_{t'}\right) \rvert = 321$)	0 ($\lvert \left(\tau''_{t''}\right) \rvert = 323$)
315	9	2 ($\lvert \left(\tau'_{t'}\right) \rvert = 323$)	3 ($\lvert \left(\tau''_{t''}\right) \rvert = 324$)
316	9	3 ($\lvert \left(\tau'_{t'}\right) \rvert = 324$)	2 ($\lvert \left(\tau''_{t''}\right) \rvert = 326$)
351	16	8 ($\lvert \left(\tau'_{t'}\right) \rvert = 366$)	2 ($\lvert \left(\tau''_{t''}\right) \rvert = 373$)
421	40	24 ($\lvert \left(\tau'_{t'}\right) \rvert = 460$)	15 ($\lvert \left(\tau''_{t''}\right) \rvert = 483$)
562	122	145 ($\lvert \left(\tau'_{t'}\right) \rvert = 683$)	275 ($\lvert \left(\tau''_{t''}\right) \rvert = 827$)
843	373	870 ($\lvert \left(\tau'_{t'}\right) \rvert = 1215$)	1997 ($\lvert \left(\tau''_{t''}\right) \rvert = 2084$)
1405	1001	2385 ($\lvert \left(\tau'_{t'}\right) \rvert = 2405$)	4789 ($\lvert \left(\tau''_{t''}\right) \rvert = 4789$)

Parameter	Wert
b	89
n	3560
c	40
v_{pad}	2
$L_{\max}$	2

398 402 402 403 ; $m_{T_{\max}} = 401.25$; $s_{T_{\max}} = 1.92$; $m_{T_{\max} \cdot c} = 16050.00$

Übersicht der Fehlerzahlen eines der Läufe:

$\lvert (\tau_t) \rvert$	$e\,(L=0)$	$e\,(L=1)$	$e\,(L=2)$
1	0		
10	0		
55	0		
280	1	0 ($\lvert \left(\tau'_{t'}\right) \rvert = 280$)	
351	4	0 ($\lvert \left(\tau'_{t'}\right) \rvert = 354$)	
386	6	2 ($\lvert \left(\tau'_{t'}\right) \rvert = 391$)	0 ($\lvert \left(\tau''_{t''}\right) \rvert = 392$)
395	8	2 ($\lvert \left(\tau'_{t'}\right) \rvert = 402$)	0 ($\lvert \left(\tau''_{t''}\right) \rvert = 403$)
400	8	2 ($\lvert \left(\tau'_{t'}\right) \rvert = 407$)	0 ($\lvert \left(\tau''_{t''}\right) \rvert = 408$)
402	8	2 ($\lvert \left(\tau'_{t'}\right) \rvert = 409$)	0 ($\lvert \left(\tau''_{t''}\right) \rvert = 410$)
403	8	2 ($\lvert \left(\tau'_{t'}\right) \rvert = 410$)	0 ($\lvert \left(\tau''_{t''}\right) \rvert = 411$)
404	8	3 ($\lvert \left(\tau'_{t'}\right) \rvert = 411$)	2 ($\lvert \left(\tau''_{t''}\right) \rvert = 413$)
421	14	6 ($\lvert \left(\tau'_{t'}\right) \rvert = 434$)	2 ($\lvert \left(\tau''_{t''}\right) \rvert = 439$)

562	61	52 ($\lvert \left(\tau'_{t'}\right) \rvert = 622$)	54 ($\lvert \left(\tau''_{t''}\right) \rvert = 673$)
843	262	550 ($\lvert \left(\tau'_{t'}\right) \rvert = 1104$)	1405 ($\lvert \left(\tau''_{t''}\right) \rvert = 1653$)
1405	844	2211 ($\lvert \left(\tau'_{t'}\right) \rvert = 2248$)	4458 ($\lvert \left(\tau''_{t''}\right) \rvert = 4458$)

Parameter	Wert
b	89
n	5340
c	60
v_{pad}	2
$L_{\max}$	2

571 574 575 580 ; $m_{T_{\max}} = 575.00$; $s_{T_{\max}} = 3.24$; $m_{T_{\max} \cdot c} = 34500.00$

Übersicht der Fehlerzahlen eines der Läufe:

$\lvert (\tau_t) \rvert$	$e\,(L=0)$	$e\,(L=1)$	$e\,(L=2)$
1	0		
10	0		
55	0		
280	0		
562	6	1 ($\lvert \left(\tau'_{t'}\right) \rvert = 567$)	0 ($\lvert \left(\tau''_{t''}\right) \rvert = 567$)
580	12	3 ($\lvert \left(\tau'_{t'}\right) \rvert = 591$)	0 ($\lvert \left(\tau''_{t''}\right) \rvert = 593$)
581	13	6 ($\lvert \left(\tau'_{t'}\right) \rvert = 593$)	1 ($\lvert \left(\tau''_{t''}\right) \rvert = 598$)
582	14	11 ($\lvert \left(\tau'_{t'}\right) \rvert = 595$)	6 ($\lvert \left(\tau''_{t''}\right) \rvert = 605$)
583	14	5 ($\lvert \left(\tau'_{t'}\right) \rvert = 596$)	3 ($\lvert \left(\tau''_{t''}\right) \rvert = 600$)
585	16	15 ($\lvert \left(\tau'_{t'}\right) \rvert = 600$)	14 ($\lvert \left(\tau''_{t''}\right) \rvert = 614$)
589	18	18 ($\lvert \left(\tau'_{t'}\right) \rvert = 606$)	6 ($\lvert \left(\tau''_{t''}\right) \rvert = 623$)
598	24	13 ($\lvert \left(\tau'_{t'}\right) \rvert = 621$)	15 ($\lvert \left(\tau''_{t''}\right) \rvert = 633$)
633	35	22 ($\lvert \left(\tau'_{t'}\right) \rvert = 667$)	31 ($\lvert \left(\tau''_{t''}\right) \rvert = 688$)
703	52	48 ($\lvert \left(\tau'_{t'}\right) \rvert = 754$)	92 ($\lvert \left(\tau''_{t''}\right) \rvert = 801$)
843	95	191 ($\lvert \left(\tau'_{t'}\right) \rvert = 937$)	477 ($\lvert \left(\tau''_{t''}\right) \rvert = 1127$)
1405	574	1807 ($\lvert \left(\tau'_{t'}\right) \rvert = 1978$)	3784 ($\lvert \left(\tau''_{t''}\right) \rvert = 3784$)

Wir erhalten in Abbildung 3.17 und Abbildung 3.18 die einspeicherbaren Assoziationen T sowie die Zahl der Zeichen Z, welche den erwarteten Verlauf zeigen, wobei die Zeichenzahl in diesem Bereich quadratisch mit der Zeilenlänge zu wachsen scheint. In Anbetracht realer Texte mit Zeilenlängen von 60-80 Zeichen, zum Beispiel auch für natürlichsprachliche Suchanfragen, ist die Ausbeute jedoch noch etwas zu gering.

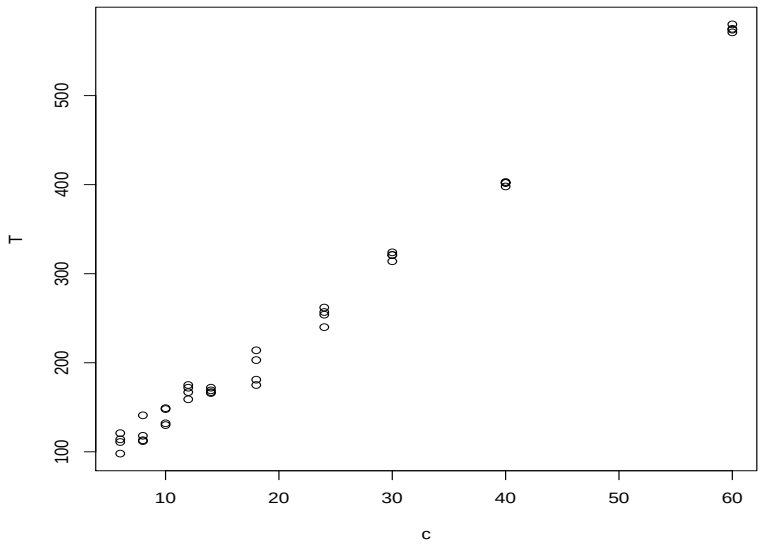


Abbildung 3.17. *Speicherbare Zeilen bei unterschiedlichen Zeilenlängen*

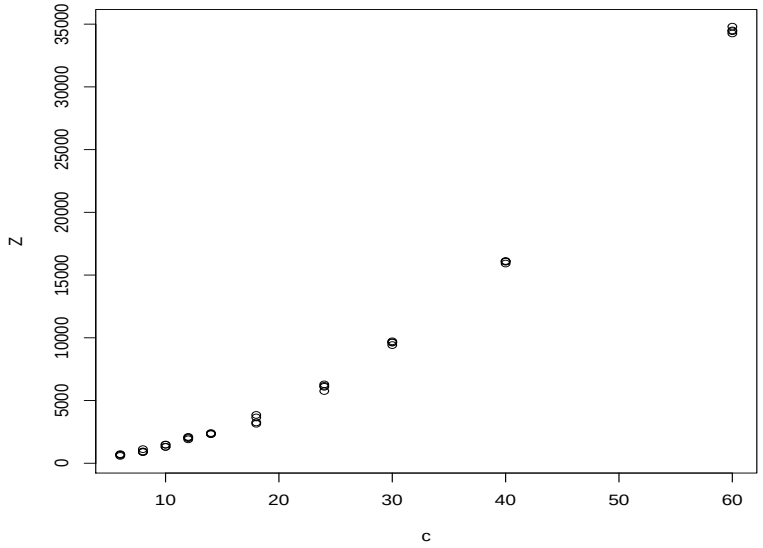


Abbildung 3.18. *Speicherbare Zeichen bei unterschiedlichen Zeilenlängen*

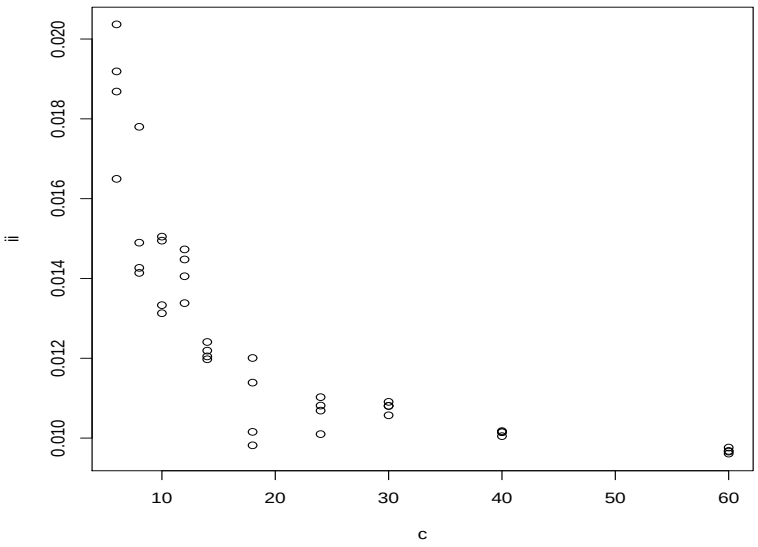


Abbildung 3.19. *Speicherbare Bits $8Z/n^2$ (Matrixgrösse)*

Dabei erhalten wir bei $c = 6$ einmal 3646 Einsen, also bei z.B. konstant 10 Bit/Eins etwa $i = 0.118733$ Speicherausnutzung und bei $c = 60$ nun 1211317 Einsen und bei 12 Bit/Eins nur noch $i \approx 0.015427$, was verbesserungswürdig ist.

3.5.4. Kritik

Als wesentlicher Kritikpunkt muss genannt werden, dass nicht alle Texte so gespeichert werden können: Wenn ein Text im Extremfall nur aus ein- und demselben Zeichen in vielfacher Wiederholung besteht, so kann die Zerlegung „keine neuen Informationen“ aus dem expandierenden Kontext des Backlogs nutzen.
Deswegen sollten auch die Strategie, immer dasselbe Zeichen als Füllzeichen zu verwenden (Abschnitt 3.5.3 Punkt 1), oder zu restriktives r (nach 2.) vermieden werden.
Überhaupt ist ein hinreichend grosses Alphabet erforderlich, um bei freier Wahl der Füllzeichen genügend verschiedene Fortsetzungen generieren zu können.
Es könnte hier empfohlen werden, die Texte vorher zu komprimieren. Hierdurch würden allerdings die Durchsuchbarkeit und auch das Prinzip der Einfachheit der Konstruktion gestört.

Für „reale Texte“ plausibler Länge spielt dies jedoch keine Rolle. Wenn mit entarteten Texten zu rechnen ist, so kann man

- entweder eine Lauflängenkodierung vornehmen, welche mittels eines Fluchtsymbols im Text untergebracht werden kann (was noch nicht gegen lange Sequenzen von <**ABABABAB**> hilft),
- oder für genügend Entropie in der Menge der Füllzeichenketten sorgen. Eine Vergrösserung der Zeilenlänge ist hier das offensichtliche Mittel der Wahl, indem es ein grösseres v_{pad} ermöglicht.

Während für reale Texte sich dieses Problem nicht ergibt, ist bei der möglichen Konstruktion eines *assoziativen Dateisystems* andererseits zu bedenken, dass hier wohl davon auszugehen sein darf, allein schon wegen der physikalischen Realisierung die Daten in nicht übermässig grossen Blöcken zu behandeln. In diesem Fall wird die zweite Variante – unter Verlust von Kapazität – funktionieren.

3.6. Kodierungen: Positionen und Nachbarschaften

Um nun die Speichermatrix gegenüber den vorigen Versuchen etwas besser ausnutzen zu können, können wir – ansonsten die beschriebene Methode beibehaltend – die „Randbedingungen des Textes“ etwas modifizieren, denn die Belegungsdichte der Merkmalsvektoren ist mit etwa 1% resultierend aus dem Umfang des Alphabetes von $b = 89$ „noch hoch“ und nicht im gewünscht spärlichen Bereich von ≤ 0.001.

3.6.1. Positionskodierungen

Wir behalten zunächst das Prinzip der Positionskodierung bei, vergrössern jedoch das Alphabet im Umfang.

Umkodierungen in ein grösseres Alphabet – Tupel ohne Überlappungen

In einem ersten Versuch könnte man mit 2-Tupeln aus dem ursprünglichen Alphabet arbeiten. Allgemein ergibt sich bei m-Tupeln über einem Alphabet von b Zeichen ein grösseres *neues* Alphabet aus $b' = b^m$ Zeichen.
Allerdings wird die Matrix schon für $m = 2$ dann sehr gross, sofern man

- ja weiterhin mindestens 6 (der jetzt „neuartigen“ Zeichen aus dem modifizierten *Pseudo-Alphabet* *Pseudo-Alphabet*

  <**(AA)**> <**(AB)**> <**(AC)**> <**(AD)**> ... <**(ZZ)**> ... [59]

  pro Zeile *und*

- die innertextliche Sequenzbildung, welche wegen des *forgesetzten Assozierens* dasselbe Format der Frage- und Antwortvektoren, und damit eine quadratische Matrix der Form $n = |F^t| = |A^t|$ erfordert,

verwenden möchte. Die Matrix hätte dann bereits Matrixgrösse

$$n = b^2 \cdot c = 89^2 \cdot 6 = 47526 \rightarrow n^2 \approx 2G$$

Einträge; mit der platzeffizienten Feld-Feld-Darstellung [60] ist dies technisch zu lösen, wenn dadurch auch die Simulation in der derzeitigen VidAs-Architektur nicht mehr möglich [61] sein wird. Aus Speichereffizienzgründen

[59] <**(AC)**> z.B. ist hier *ein Zeichen* des (Alphabetes des) transformierten Textes; das Alphabet ist nur angedeutet dargestellt: es besteht aus (hier) *allen* 89^2 *Kombinationen* zweier beliebiger Zeichen aus dem originalen Text, also ebenfalls von *Kleinbuchstaben* und *Sonderzeichen* incl. Zeilenumbrüchen, welche im Originaltext auftreten. *Nicht* besonders beachtet wird dabei, dass *nicht alle Kombinationen auch tatsächlich auftreten*

[60] [Hagström 96] Kap. 3.1.3

[61] bei Realisierungen z.B. als *FPGA* einfacher Bauart; es ist hier aufgrund der Voll-Darstellung der Matrix durch logische Schaltkreise von einer Grösse bis zu maximal $n = 500$ auszugehen

wollen wir auf die Möglichkeit der unkomprimierten Darstellung deswegen verzichten.
Wir wollen dies jetzt kontrollieren, und anschliessend einen weniger extrem skalierenden Ansatz noch darstellen.
Es sind zunächst (zur Platzersparnis horizontal) die Ergebnisse bei Verwendung von 2-Tupeln *ohne reblocking* dargestellt: Aufgrund der verringerten Kollisionswahrscheinlichkeiten ist im letzten Test bei $c = 20$ hier zuerst *ohne reblocking* bereits $2 \cdot m_{T_{max} \cdot c} = 40560$ an speicherbaren Zeichen zu erreichen – dies ist zu vergleichen mit $c = 40$ auf Seite 70, resultierend in 16050 Zeichen, dort *mit reblocking*.

$T_{\max} - 1 = 186$

Parameter	Wert
b	7921
n	47526
c	6

$\lvert(\tau_t)\rvert$	1	10	55	168	182	186	187	188	189	196	224	280
$e\,(L=0)$	0	0	0	0	0	0	1	1	1	3	9	12

$T_{\max} - 1 = 143$

Parameter	Wert
b	7921
n	63368
c	8

$\lvert(\tau_t)\rvert$	1	10	55	112	140	142	143	144	147	154	168	280
$e\,(L=0)$	0	0	0	0	0	0	0	2	2	2	4	5

$T_{\max} - 1 = 142$

Parameter	Wert
b	7921
n	79210
c	10

$\lvert(\tau_t)\rvert$	1	10	55	112	140	142	143	144	147	154	168	280
$e\,(L=0)$	0	0	0	0	0	0	1	1	1	1	1	1

$T_{\max} - 1 = 1146$

Parameter	Wert
b	7921
n	118815
c	15

$\lvert(\tau_t)\rvert$	1	10	55	280	843	1124	1142	1145	1146	1147	1151	1160	1195	1265	1405
$e\,(L=0)$	0	0	0	0	0	0	0	0	0	1	2	3	12	16	18

Parameter	Wert
b	7921
n	158420
c	20

$T_{\max} - 1 = 1014$

$\lvert(\tau_t)\rvert$	1	10	55	280	843	984	1002	1011	1013	1014	1015	1019	1054	1124	1405
$e\,(L=0)$	0	0	0	0	0	0	0	0	0	0	1	1	1	1	3

Da sich die Fehlerzahlen in den bisherigen Untersuchungen als hinreichend "gutartig" erwiesen haben in dem Sinne, dass keine extremen Ausreisser festgestellt wurden, und für unsere grösseren und noch spärlicheren Matrizen günstigerweise die Kollisionswahrscheinlichkeiten noch abnehmen, belassen wir es im folgenden ohne nähere Prüfung bei jeweils einem Versuch, wenn wir nun die 2-Tupel-Variante der Kodierung der Zeileninhalte mit der maximal 2-stufigen „Aufteilen-und-Auffüllen"-Vorverarbeitung kombinieren:

$T_{\max}$ 3146

Parameter	Wert
b	7921
n	79210
c	10
v_{pad}	2
$L_{\max}$	2

$\lvert(\tau_t)\rvert$	$e\,(L=0)$	$e\,(L=1)$	$e\,(L=2)$
1	0		
10	0		
55	0		
280	1	0 ($\lvert(\tau'_{t'})\rvert = 280$)	
1405	3	0 ($\lvert(\tau'_{t'})\rvert = 1407$)	
2812	82	0 ($\lvert(\tau'_{t'})\rvert = 2893$)	
2988	86	0 ($\lvert(\tau'_{t'})\rvert = 3073$)	
3076	93	0 ($\lvert(\tau'_{t'})\rvert = 3168$)	
3120	95	0 ($\lvert(\tau'_{t'})\rvert = 3214$)	
3142	95	0 ($\lvert(\tau'_{t'})\rvert = 3236$)	
3145	95	0 ($\lvert(\tau'_{t'})\rvert = 3239$)	
3146	95	0 ($\lvert(\tau'_{t'})\rvert = 3240$)	
3147	95	2 ($\lvert(\tau'_{t'})\rvert = 3241$)	1 ($\lvert(\tau''_{t''})\rvert = 3242$)
3148	95	2 ($\lvert(\tau'_{t'})\rvert = 3242$)	1 ($\lvert(\tau''_{t''})\rvert = 3243$)
3153	95	2 ($\lvert(\tau'_{t'})\rvert = 3247$)	1 ($\lvert(\tau''_{t''})\rvert = 3248$)
3164	96	1 ($\lvert(\tau'_{t'})\rvert = 3259$)	1 ($\lvert(\tau''_{t''})\rvert = 3259$)
3515	116	4 ($\lvert(\tau'_{t'})\rvert = 3630$)	1 ($\lvert(\tau''_{t''})\rvert = 3633$)
4218	162	14 ($\lvert(\tau'_{t'})\rvert = 4379$)	3 ($\lvert(\tau''_{t''})\rvert = 4392$)
7030	864	137 ($\lvert(\tau'_{t'})\rvert = 7893$)	25 ($\lvert(\tau''_{t''})\rvert = 8029$)

$T_{\max}$ 5435

Parameter	Wert
b	7921
n	118815
c	15
v_{pad}	2
$L_{\max}$	2

$\lvert(\tau_t)\rvert$	$e\,(L=0)$	$e\,(L=1)$	$e\,(L=2)$
1	0		
10	0		
55	0		
280	0		
1405	18	0 ($\lvert(\tau'_{t'})\rvert = 1422$)	
5435	74	2 ($\lvert(\tau'_{t'})\rvert = 5508$)	0 ($\lvert(\tau''_{t''})\rvert = 5509$)

Parameter	Wert
b	7921
n	158420
c	20
v_{pad}	2
$L_{\max}$	2

$T_{\max}$ 4076

$\lvert(\tau_t)\rvert$	$e\,(L=0)$	$e\,(L=1)$
1	0	
10	0	
55	0	
280	0	
1405	3	0 ($\lvert\left(\tau'_{t'}\right)\rvert = 1407$)
4076	9	0 ($\lvert\left(\tau'_{t'}\right)\rvert = 4084$)

$T_{\max}$ 3261

Parameter	Wert
b	7921
n	198025
c	25
v_{pad}	2
$L_{\max}$	2

$\lvert(\tau_t)\rvert$	$e\,(L=0)$	$e\,(L=1)$
1	0	
10	0	
55	0	
280	0	
1405	0	
3261	1	0 ($\lvert\left(\tau'_{t'}\right)\rvert = 3261$)

$T_{\max}$ 2717

Parameter	Wert
b	7921
n	237630
c	30
v_{pad}	2
$L_{\max}$	2

$\lvert(\tau_t)\rvert$	$e\,(L=0)$
1	0
10	0
55	0
280	0
1405	0
2717	0

Im wesentlichen wird hier ersichtlich, dass schliesslich schon mit $c'' = 15$, was ja $c = 30$ bezogen auf den Originaltext entspricht, ein überragendes Ergebnis erzielt werden kann: Mit einer Kapazität von $T \cdot c \cdot 2$ (Faktor 2 wegen Umkodierung) $= 5435 \cdot 15 \cdot 2 = 163050$ eingespeicherten Zeichen kann der gesamte Text eingeschrieben werden.
Wird die Zeilenlänge noch geringfügig erhöht ($c'' = 20$), so treten bei Einschreibung des gesamten Textes nur 9 Fehler in $L = 0$ auf, und es bedarf sogar nur der Korrektur durch l_1 allein; die zusätzliche Sicherheit durch den vorhandenen Mechanismus von l_2 schadet natürlich nicht.

Ergebnis des Umkodierens Nachdem es mit Hilfe

(l_1): der Umkodierung zur Verringerung der Bitdichte und

(l_2, l_3): der (dann immer noch notwendigen, hier zweistufigen) Transformation des Textes zur Disambiguierung ähnlicher Sequenzen durch *Aufteilen und Auffüllen*

jetzt möglich wurde, den gesamten Zieltext in einen Matrixspeicher einzuschreiben und wieder auszulesen, ist Gelegenheit, einige Anmerkungen zu tätigen beziehungsweise weiterführende Fragen anzudeuten:

- Noch mehr Korrekturebenen lohnen nicht; man sieht zum Teil auch, dass in den Fällen, wo viele Fehler zu korrigieren waren, schlechte Ergebnisse erzielt werden
- das Verfahren eignet sich also besser für die Detailkorrekturen der extrem spärlichen Modelle,
- die komprimierte Speicherung nach der Darstellung von [Hagström 96] ist für die Möglichkeit perfekten retrievals und damit die Textspeicherung als solche offenbar entscheidend.
- Das Problem, den Anfangswert (also die erste Zeile oder Frage F^0) zu speichern beziehungsweise bei einer Sendung der Matrix „mitzuteilen“, ist im Detail noch nicht betrachtet,
- aber der Umfang der Segmente ist dagegen *nicht* so gross, dass man sagen würde, man hätte einen relevanten Teil des Problems der Textspeicherung umgangen [(62)]

Von-m-zu-q-Tupel-Bildung Das beschriebene Verfahren lässt sich wie angedeutet verallgemeinern, und damit bezüglich der extremen Skalierung der Matrixgrösse abschwächen falls gewünscht, indem zunächst wieder m Zeichen zu einem *Pseudo-Zeichen* eines Alphabetes C' zusammengefasst werden, dann jedoch dieses wieder in q Zeichen, also ein q-Tupel über einem weiteren Alphabet C'', aufgeteilt wird, dessen Umfang dann

$$|C''| = \left\lceil |C|^{\frac{m}{q}} \right\rceil$$

beträgt:

$$(\zeta_1, \ldots, \zeta_m) \rightarrow (\zeta'_1) \rightarrow (\zeta''_1, \ldots, \zeta''_q) \quad \zeta_i \in C,\, \zeta'_i \in C',\, \zeta''_i \in C''$$

[(62)] dies wäre zum Beispiel dann der Fall, wenn wir einen Text der Länge Z in 2 Zeilen (F^0 und F^1) der Länge $c = \frac{Z}{2}$ aufteilten, von denen wie oben gesagt also tatsächlich 50% der Nutzdaten „auf anderem Wege“ verarbeitet würden

Eine Verringerung der Bitdichte für dieselbe Textmenge ergibt sich [63] immer dann, wenn die Skalierung der Zeichen-Anzahl des Textes $u := \frac{q}{m} > 1$ ist; dies gilt auch wenn so möglicherweise noch mehr als 2 der ursprünglichen Zeichen dann jedoch auch wieder in *mehr als ein* „Pseudo-Zeichen“ eines dann *nicht ganz so umfangreichen* Alphabetes umgewandelt werden.

3.6.2. Verkleinerung der Matrixgrösse durch Transformationen

Eine weitere Verfahrensmodifikation ergibt sich, wenn man die Zeilenlänge c reduziert, beziehungsweise das Suchverfahren umkehrt, indem zu einem vorgegebenen Text der Länge Z tatsächlich doch zunächst mit $c = \frac{Z}{2}$ angesetzt und dann die Zeilenlänge sukzessive verringert wird, bis keine Speicherung mehr möglich ist. Der gegebenenfalls „grosse“ Resttext F^0 kann mit derselben Methode behandelt werden, wenn mehrere oder gar beliebig viele Speicher zur Verfügung stehen, wie in einer virtuellen Implementierung.
Eine Basis der freien Parameter des erarbeiteten Kodiermodells bilden zum Beispiel, c'', $b'' = |C''|$ zusammen mit v_{pad}, r.
Dabei legen die ersten beiden die Dimension und Belegungsdichte der Matrix fest. Ein weiterer „möglicherweise“ freier Parameter ist in diesem Zusammenhang allerdings noch die Maschinenkonstante α, welche die Anzahl der Bits zur physikalischen „Speicherung der Eins“, also in der Praxis der Adresse einer ihrer Dimensionen bezeichnet.
Die weiteren beiden Parameter sind der Abwechslungsgrad der Fülltexte zwischen und innerhalb seiner einzelnen Positionen und bestimmen damit die Entropiebeimischung zur Disambiguierung, wobei der eine Lokalisations- und der zweite Abstandscharakter hat.

3.7. Diskussion

3.7.1. Lernen modifizierter Texte oder iteratives Retrieval?

Fehlervermeidende Verfahren, bei denen also zum Beispiel die Kodierung der *Fragen* und *Antworten* sich wie dargestellt ändern kann, können sich für bestimmte Aufgaben als vorteilhaft erweisen.
Im Zusammenhang mit der spärlichen Matrixdarstellung treten nur *selten* Fehler auf, welche *einfach* korrigiert werden können.
Möglicherweise muss jedoch beim Einspeichervorgang – zum Teil mehrfach rückwärts – die Kodierung von Datensätzen oder Aufteilung in Datensätze nachträglich angepasst werden.
Da Matrixspeicher jedoch die Assoziationen nur kumulieren, erfordert dies ein völliges Neulernen oder eine optimierte Verwaltungsstruktur mit zwischengespeicherten Zuständen. Dieses ist zumindest ein Ausschlusskriterium bei

[63] „in der Praxis“; Grenzfälle und dass sich der Text natürlich nicht unbedingt in m-Tupel ohne Rest aufteilen lässt, ignorieren wir hier

möglichst einfacher Implementierung komplett in Hardware, wenn nicht komplizierte Kaskaden von Speichermodulen in grösserer Anzahl aufwendig verschaltet und gesteuert werden.
Für Archivierungs-, Speicher- oder schnelle Suchaufgaben ist dieses jedoch nicht unbedingt ein Nachteil.

3.7.2. Speicheraufwand

Wo es jetzt mit dieser Methode gelungen ist, einen nicht-trivialen, realen Text in den Matrixspeicher einzuschreiben und wieder auszulesen, stellt sich abschliessend die Frage, ob hier überhaupt eine relevante Leistung im Sinne assoziativer Speicherung vorliegt.
Wie in den anfänglichen Beispielen dargelegt, ist bei hinreichend grosser Matrix davon auszugehen, dass das Verfahren zu einem quasi-herkömmlichen verstreuten Speichern (ohne relevante Nutzung der Überlagerungseigenschaften) degeneriert.
Während für „relevante Nutzung“ eine formale Definition nur schwer angebbar ist, lässt sich anhand des zur Verwaltung des Memory-Inhaltes notwendigen Speicheraufwandes – der in der Simulation verwendeten Feld-Feld-Darstellung – abschätzen, ob sich das Modell im günstigen Bereich assoziativer Speicherung bewegt:
Nach [Hagström 96] Kapitel 3.1.3 ist die zugehörige Optimalitätsbedingung hier:

$$-\frac{\ln p}{p} \approx T\frac{\ln T}{\alpha \ln 2} \tag{O}$$

mit z.B. $\alpha = 32 \approx ld\, n$. Mit unseren Werten aus dem Lauf auf Seite 78 ergibt sich für $p = \frac{20}{158420}$, $T = 4076$:

$$70k \approx 1.5k \tag{o}$$

Dies scheint auf den ersten Blick noch nicht gut, jedoch sind laut [Hagström 96] ± 1.5 Grössenordnungen unkritisch, und für $p' = 15 \cdot p$ erhalten wir schon eine brauchbare Annäherung.
Als zugehörige Ausnutzung ergibt sich:

$$H_{FF}(n, p = \frac{k}{n}, T) = \frac{T \cdot n(p\ ld\, p + q\, ld\, q)}{n \cdot \alpha + npT\, ld\, T} \tag{H}$$

≈ 0.6, was uns zufriedenstellt.

3.7.3. Anmerkung zur Implementierung

Was den Auslesevorgang angeht, so stellt sich noch die Frage, ob dieser einfach auch in Hardware implementiert werden kann, um eine Verbindung mit Komponenten des VidAs-Systems zu ermöglichen. Dies ist jedoch relativ einfach möglich:

Zwischenspeicher Wenn man $L_{\max}$, wie in den Beispielen demonstriert, klein hält, kann man die Rekonstruktion der ursprünglichen $\{\tau_t\}$ aus den $\{\tau^{(l)}_{t^{(l)}}\}$ mit insgesamt $K = 2^{L_{\max}+1} - 1$ den in VidAs definierten Registern ähnlichen Zwischenspeichern leicht realisieren. Diese sind logisch in Binärbaumstruktur anzuordnen und bilden implizit den notwendigen Teil des *Backlog* ($B_{i,k\leq K}$). Die Wurzel auf Ebene $l = 0$ ist dabei das „Ausgaberegister“ für zusammengesetzte (oder von Anfang an vollständige) Vektoren.

Zusammensetzen Zwischen den Registern müssen nur einfache Kopieraktionen von zwei Hälften der Register der Ebene l in das übergeordnete Register der Ebene $l - 1$ implementiert werden.

Erkennen Zu erkennen ist, ob in linker oder rechter Hälfte (je nachdem um welches Register es sich handelt) maximal eine bestimmte Anzahl verschiedener Zeichen kodiert vorliegt. Bei der „auf Platz“-Kodierung ist dies möglich, indem alle einem bestimmten Zeichen entsprechenden Positionen durch logische Schaltung verodert werden. Über diese b Stück Ausgänge ist dann eine Summe zu bilden.
Falls man hier an Schaltungsaufwand sparen möchte, kann man auch bestimmte Forderungen an die Ersatzzeichenfolgen stellen. Wenn zum Beispiel bei der Konstruktion der Fülltexte die verschiedenen Zeichen zyklisch gesetzt werden, reicht es auch aus, immer nur lokale Addierer vom Umfang der v_{pad} zu implementieren, deren Ausgänge dann Und-verknüpft werden.

Ablaufsteuerung Die Steuerung des Prozesses ist auf oberster Ebene mit einem endlichen Automaten mit Z Zuständen möglich, welche abbilden, in welchen Zwischenspeicher der nächste Antwortvektor einzutragen ist. Dann ist jeweils zu prüfen, ob dieser wieder die verallgemeinerte Dekodierungsbedingung nach $P2'$ erfüllt und entsprechend entweder auf weitere Vektoren in Speichern nächster Ordnung „zu warten“ oder eine Kaskade des Zusammensetzens in Richtung zum Ausgabespeicher hin auszulösen.

Füllzeichen Um bei beliebigen Texten die anfangs geforderte Randbedingung, dass nicht mehr als $Q = c/2$ gleiche Zeichen aufeinander folgen dürfen, sicherstellen zu können, wurde vorgeschlagen, das Alphabet C um ein *Füllzeichen* zu erweitern, welches notfalls an beliebiger Stelle in solchen Sequenzen eingefügt werden kann und bei der Ausgabe ignoriert werden muss.
Sollen die Texte innerhalb des Systems anders weiter verarbeitet werden, so ist es am einfachsten, das Füllzeichen an genau der Hälfte aller Positionen einzufügen, um zum Entfernen auch den soeben beschriebenen Mechanismus wieder verwenden zu können.

4. Anknüpfungspunkte

4.1. Mögliche Erweiterungen

Zur Verbesserung des allgemeines Verfahrensablaufs der Speicherung (und des Auslesens) von Texten kann man möglicherweise an einigen der bisher nicht näher betrachteten Punkten anknüpfen:

- In Abschnitt 3.4.2 betrachteten wir die Option, durch Ausprobieren unter Kenntnis von Eigenschaften der verwendeten Kodierung wieder „in die Textsequenz zu finden".

  Es wäre, am einfachsten wohl durch Simulationen, zu prüfen, unter welchen Randbedingungen dies in der überwiegenden Zahl der Fälle mit zu definierendem Aufwand möglich ist.

4.2. Antworten

Besser beantworten zum beschriebenen Verfahren lassen sich inzwischen auch die Fragen:

- Q: Kann man (wie beim Aufschlagen eines Buches) auch irgendwo „in der Mitte" (weiter-)lesen? (Dies steht natürlich auch im direkten Zusammenhang mit der Durchsuchbarkeit des Textspeichers.)

  A: Durch Anfragen eines hinreichend grossen Teilstückes (sicherlich reichen $2c''$ Pseudozeichen) des Textes in maximal c'' verschobenen Anfragen kann mit einer in den Beispielen hohen Wahrscheinlichkeit (in den einzelnen Tabellen kann dies als $p = 1 - \frac{e(L=0)[|\{\tau_t\}=T_{\max}|]}{T_{\max}}$ für den konkreten Fall abgelesen werden), so zum Beispiel im Falle des kurzen $c = 10$ auf S. 78 mit $\approx 98.9\%$. Sofern deterministisch, können (l_1) beziehungsweise (l_2) dabei offensichtlich auch angewandt werden, um iterativ zu einem *recall* von 1 zu gelangen.

  Bei nicht-deterministischen l' sind bei 2-Tupeln schlimmstenfalls $b^{v_{pad}}$ dazu zu prüfen, wobei hier die *precision* nicht mehr 1 sein muss.

- Q: Findet man *überhaupt* die t oder „nur" die Menge aller $\{A^t\}$ oder $\{F^t\}$, welche dann zur Rekonstruktion von T ausreichen?

  A: Im präsentierten Algorithmus lassen sich ohne besonderen Aufwand die τ_t in der gewünschten Ordnung auslesen.

4.3. Exkurs: Lange Sequenzen

Während unserer Untersuchung an *kleinen* Assoziativspeicher sind uns überraschend lange Assoziationsketten begegnet. Es handelt sich also um besonders kompakt konstruierte kombinatorische Sequenzen.
Diese Assoziationsketten können brauchbar sein, um in besonders kompakten Matrizen Sequenzen abzulegen, zum Beispiel für Ablaufsteuerzwecke als Fortsetzungskern bei kleinen Matrizen den Daten zur Ordnung beigefügt zu werden.
Teilweise wurden diese in [Töpfer 11] gefunden, andere – auch systematischere – haben wir selbst wieder mit dem Programm *igehirn.py* zusammengetragen und hier dargestellt.

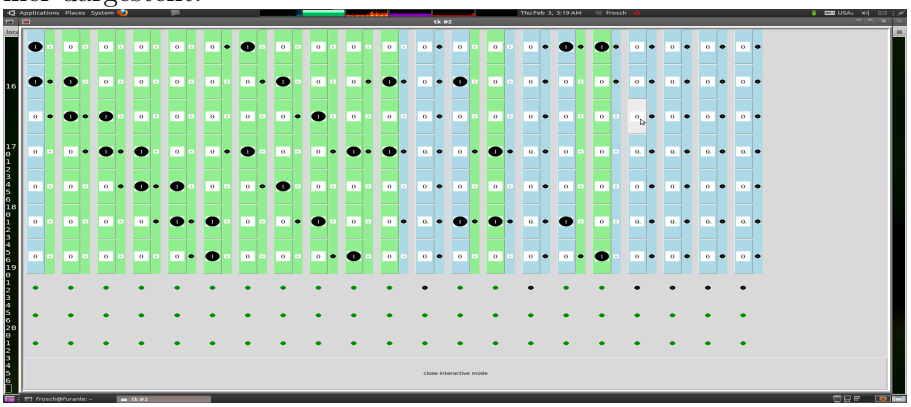


Abbildung 4.1. *Lange Sequenz in 7x7-Matrix*

Beispiele sind hier: Eine 10-schrittige Assoziationskette mit detektierbarem Ende (Abbildung 4.1), eine 7-schrittige Assoziationskette in einer kleinen (6x6) Matrix mit detektierbarem Ende (Abbildung 4.2),

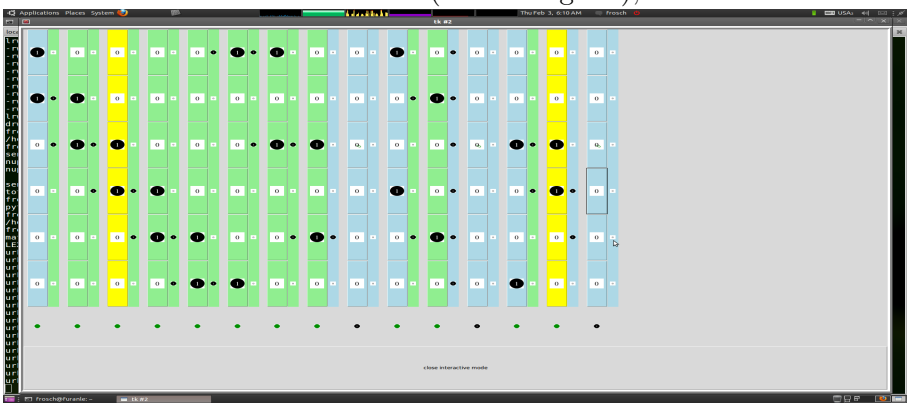


Abbildung 4.2. *Lange Sequenz in 6x6-Matrix*

sowie eine zunächst 12-schrittige Assoziationskette, welche dann (unendlich) in sich selbst mündet (Abbildung 4.3), indem der 13. Vektor wieder in den achten übergeht.

Abbildung 4.3. *Lange Sequenz in 7x7-Matrix – die roten Markierungen haben hier keine Bedeutung*

In denselben Matrizen finden natürlich oft auch noch weitere Assoziationsketten Platz, wobei wir hier immer nur diejenigen mit identischer Bitdichte betrachten – was natürlich nicht allgemein beim Kodieren eingehalten werden muss.
Wir wollen noch kurz zeigen, wie diese Sequenzen zur Datenspeicherung beziehungsweise Ablaufsteuerung genutzt werden können. Dazu mögen im folgenden Bild 4.4 die unteren sieben Zeilen von unten nach oben die Buchstaben eines kleinen Alphabetes <**ACINOST**> bedeuten – sie können natürlich auch der Ansteuerung von Aktoren einer Maschine dienen.

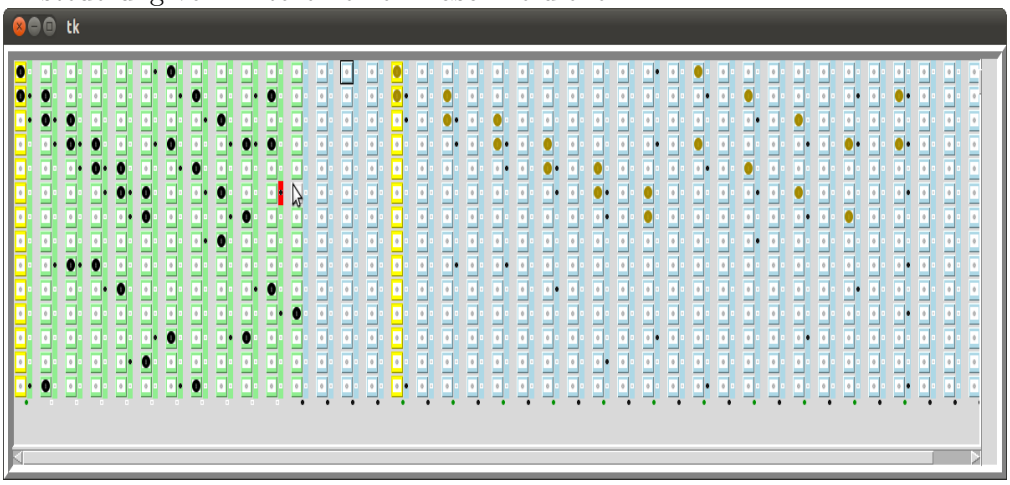


Abbildung 4.4. *Ein durch eine „lange Sequenz“ gespeicherter Text*

Im linken Teil der Abbildung wird oben zum Lernen dieselbe Sequenz wie in Bild 4.1 verwendet, um unten eine Sequenz ablegen zu können. Diese ist dabei um einen Schritt versetzt, da der erste Buchstabe des Output-Wortes anfangs beim Auslesen noch nicht bekannt sein soll, das heisst im Gegensatz zu der im Abschnitt 3.5 geschilderten Methode trägt q_0 hier keine Textinformation. Insofern ist dieses ein Beispiel, in welchem der „Startschlüssel“ zum Dekodieren nicht mit angegeben werden muss:

Man fängt – ausgehend allein von einer konstanten Bitdichte $p_F = \frac{2}{7}$ des Fortsetzungsfadens – hier einfach mit den ersten beiden Positionen an, die Matrix zu befragen und die Buchstaben zu notieren. Im oberen Teil erhält man immer den Folgeeintrag aus der bekannten Sequenz [(1)] .
Interessant zu sehen ist aber auch, dass man „ohne die Buchstaben“ wieder anfragen kann – dies ist im rechten Teil in andersfarbigen Punkten dargestellt (beginnend mit dem gelb markierten Vektor, was ja die Wiederholung des q_0 dort anzeigt).
Durch die freie Spalte zwischen den Vektoren dort ist dies nicht ein Lern-, sondern ein reiner Abfragebereich, wo nur die einzelnen Abfrageschritte sichtbar sind und man die Output-Buchstaben in den kleinen Punkten trotzdem erkennen kann. Mit dieser 10 Assoziationen umfassenden Folge ist allerdings der letzte Buchstabe so nicht mehr eindeutig abzulesen – und dieser Zustand lässt sich auch durch keinerlei Hinzufügen von Bits im durch den Pfeil markierten Vektor über diesem Buchstaben erreichen. Allerdings kann man durch logisches Schliessen hier zu der korrekten Lösung gelangen.

4.4. Verbesserungen

Hier soll nur Platz sein, um einige Ideen zu vermerken, welche bisher nicht geprüft wurden.

Schnelleres Lernen Zur Beschleunigung des Verfahrens können Vektoren, welche sich als kritisch erwiesen haben, im Nachfolgenden durchlauf zuerst getestet und zerlegt werden, um zumindest bei der laufenden Suche nach nutzbaren Parametern wie c oder b schneller abzubrechen.

Nicht konstante Blockgrössen Eine sich spontan aufdrängende Idee ist natürlich die Verwendung der texteigenen Zeilenumbrüche oder Wortlängen.

Mehr Mischen Die Zerlegungsstrategie in linke und rechte Teile könnte verfeinert werden, zum Beispiel zu einer Verzahnung. Teilung in nicht exakte Hälften dagegen würde das Halten zusätzlicher Informationen erfordern.

Gleichzeitige Verwendung mehrerer Beschreibungen Es spricht nichts dagegen, auch noch längere Tupel zusätzlich als Eigenschaften zu vermerken. Vermutlich hilft dies jedoch mehr bei der Suche als bei der Speicherung.

Lesende Korrektur durch Aufbau eines Gesamtbildes Die Kodierungen können um zeilenübergreifende Merkmale ergänzt werden, so dass das Backlog angereichert wird. Auf diesem Wege können 2-dimensionale „Inseln“ beim

[(1)] Die zugehörige Matrix findet sich abgedruckt auf S. 101

Auslesen wie in einem Puzzle zusammengesetz werden, was im Grunde einen Spezialfall von “mehrere Beschreibungen“ darstellt. Hierzu wurden bereits in anderen Zusammenhängen vielversprechende Experimente vorgenommen.

5. Schlussbemerkungen

Wie eingangs formuliert, kann die gefundene Darstellung als einzige gespeicherte kompakte Version eines Textes dennoch schnell durchsucht werden, auch wenn nachträgliche Änderungen gar nicht oder nur mit hohem Aufwand möglich sind. Dieses Eigenschaftsprofil deckt sich besonders mit dem der Archivierung von Dokumenten in natürlicher Weise, weswegen hier die Assoziativspeicher einen neuen Aufgabenbereich finden können.
Entsprechendes gilt offensichtlich auch für die Anwendungen der Versionskontrolle sowie Deduplizierung, und es bleibt zu untersuchen, wie ein Dateisystem auf diesen Strukturen aufgebaut werden kann.
Für Textspeicherung zu derartigen Zwecken ist es auch grundsätzlich unproblematisch, einen Text in mehreren Blöcken und damit Memories zu speichen, sobald ein gegebenes Volumen nicht mehr korrekt von *einem* Memory verarbeitet werden kann.
Ebenfalls ist noch als günstig anzumerken, dass mit relativ kurzen einfach strukturierten Tupeln gearbeitet werden konnte, wodurch nicht von der zeicheneigenen Granularität abgewichen werden musste. Dies macht ein Suchen nach Inhalten unkompliziert.
Nachdem die Textspeicherung dem assoziativen Kalkül prinzipiell zugänglich gemacht wurde, können die Ergebnisse mit den dort üblichen Methoden interpretiert werden – auch um die Struktur der Nutzdaten zu analysieren.
Während schon viele Detailmodifikationen im Ausleseprozess von Assoziativspeichern in der Literatur beschrieben sind, handelt es sich hier um ein eher *äusseres Verfahren* zur Text-Kodierung nach Transformation.
Der Ausleseprozess ist modifiziert und kombiniert Ergebnisse mehrerer Anfragen an einen Speicher *auf neuartige, jedoch einfache Weise*, welche klar strukturiert ist und mit wenig Aufwand auch *in Hardware realisiert* werden kann.
Üblicherweise wird im Bereich des maschinellen Lernens mittels Modifikationen der Verfahren und Anpassung der Parameter versucht, *Recall* und *Precision* zu verbessern. Im Gegensatz dazu startet – der Aufgabe eines *Speichers* angemessen – diese Arbeit mit der bei netzartigen IR-Strukturen ungewöhnlichen Forderung „$Recall = Precision = 1$“ und versucht, dies für gegebene Texte mit möglichst einfachen Mitteln zu erreichen. So kann der Maschine eine Abfolge direkt eingegeben werden.
Welche Daten dazu umkodiert werden müssen, kann auf verschiedene Weisen bestimmt werden – möglicherweise wie vorgeschlagen durch ein autoassozitives Netz.

Die Art der rein mechanischen – in dem Sinne, dass hier keine bedeutungstragenden Einheiten *gewählt* werden müssen, sondern einfache Ersetzungsregeln verwendet werden können – Transformation ist anpassbar und führt wenn schlecht im Sinne von übermässig oft angewandt im wesentlichen nur zu einer Verlangsamung des Einspeicherprozesses.
Emergente Phänomene können innerhalb eines natürlichen wie auch KNN wohl immer auftreten – aber man sollte nicht übersehen, welche Bedeutung *ersten Worten* entwicklungshistorisch beigemessen wird.
Und diese Worte und Lautfolgen sind beim Menschen immer nur einfache und unveränderte Wiederholungen von Vorgesprochenem – nur dann können wir sie ja als korrekt wiedererkennen.
Nachdem wir dies erfolgreich modelliert haben – das heisst, beliebige Daten einem neuronalen Netz zuzuführen und unverändert wieder zurückzuerhalten – stehen die wenigen Modellparameter einer weiteren Interpretation aber auch mechanischer Verarbeitung auf anderen Ebenen zur Verfügung.
Weiterhin können dank exakter Abrufbarkeit von Zeichenketten auch Programme in Matrixspeicher eingeschrieben werden, welche sich deterministisch verhalten.

A. Als Testdaten verwendete Texte, weitere Ablaufprotokolle

A.1. Testdaten/Texte

Dateinamenliste

```
=== ZEILEN, WORTE, ZEICHEN (8-Bit) ==== 2210   7958 159903 find-160-notilde.txt ==== ANFANG (28 zeilen) ===


/home/frosch
/home/frosch/Videos
/home/frosch/.bash_history
/home/frosch/.config
/home/frosch/.config/monitors.xml.backup
/home/frosch/.config/nautilus
/home/frosch/.config/nautilus/desktop-metadata
/home/frosch/.config/user-dirs.locale
/home/frosch/.config/gnome-session
/home/frosch/.config/gnome-session/saved-session
/home/frosch/.config/gwibber
/home/frosch/.config/gwibber/gwibber.sqlite
/home/frosch/.config/dconf
/home/frosch/.config/dconf/user
/home/frosch/.config/libaccounts-glib
/home/frosch/.config/libaccounts-glib/accounts.db-wal
/home/frosch/.config/libaccounts-glib/accounts.db-shm
/home/frosch/.config/libaccounts-glib/accounts.db
/home/frosch/.config/monitors.xml
/home/frosch/.config/compiz-1
/home/frosch/.config/compiz-1/compizconfig
/home/frosch/.config/compiz-1/compizconfig/config
/home/frosch/.config/compiz-1/compizconfig/done_upgrades
/home/frosch/.config/user-dirs.dirs
/home/frosch/.config/Trolltech.conf
/home/frosch/.config/update-notifier
/home/frosch/.config/ibus
/home/frosch/.config/ibus/bus


.
.
.
```

Alice im Wunderland

```
=== ZEILEN, WORTE, ZEICHEN (8-Bit) ==== 3686  25697 163106 alice-wunderland-pg19778-easy.txt ==== ANFANG (20 zeilen) ===

Alice's Abenteuer

im Wunderland

von

Lewis Carroll.

Aus dem Englischen von Antonie Zimmermann.



Mit zweiundvierzig Illustrationen

von

John Tenniel.

Autorisirte Ausgabe.


.
.
.

=== ZEILEN, WORTE, ZEICHEN (8-Bit) ==== 3686  25697 163106 alice-wunderland-pg19778-easy.txt ==== ENDE (20 zeilen) =====

Alles wieder zur alltäglichen Wirklichkeit werden würde; das Gras würde
dann nur im Winde rauschen, der Teich mit seinem Rieseln das Wogen des
Rohres begleiten; das Klappern der Theetassen würde sich in klingende
Heerdenglocken verwandeln und die gellende Stimme der Königin in die
Rufe des Hirtenknaben -- und das Niesen des Kindes, das Geschrei des
Greifen und all die andern außerordentlichen Töne würden sich (das wußte
sie) in das verworrene Getöse des geschäftigen Gutshofes verwandeln --
während sie statt des schwermüthigen Schluchzens der falschen
Schildkröte in der Ferne das wohlbekannte Brüllen des Rindviehes hören
würde.

Endlich malte sie sich aus, wie ihre kleine Schwester Alice in späterer
Zeit selbst erwachsen sein werde; und wie sie durch alle reiferen Jahre
hindurch das einfache liebevolle Herz ihrer Kindheit bewahren, und wie
sie andere kleine Kinder um sich versammeln und _deren_ Blicke neugierig
und gespannt machen werde mit manch einer wunderbaren Erzählung,
vielleicht sogar mit dem Traume vom Wunderlande aus alten Zeiten; und
wie sie alle ihre kleinen Sorgen nachfühlen, sich über alle ihre kleinen
Freuden mitfreuen werde in der Erinnerung an ihr eigenes Kindesleben und
die glücklichen Sommertage.
```

Alice in Wonderland

```
=== ZEILEN, WORTE, ZEICHEN (8-Bit) ==== 3635  27280 157172 alice-wonderland-pg28885-easy.txt ==== ANFANG (20 zeilen) ===

ALICE'S ADVENTURES IN WONDERLAND

[Illustration: "Alice"]

[Illustration:

          ALICE'SÂ·ADVENTURES
          INÂ·WONDERLAND
          BYÂ·LEWISÂ·CARROLL
          ILLUSTRATEDÂ·BY
          ARTHURÂ·RACKHAM

          WITH A PROEM BY AUSTIN DOBSON

          LONDONÂ·WILLIAMÂ·HEINEMANN
          NEWÂ·YORKÂ·DOUBLEDAYÂ·PAGEÂ·&Â·Co]

          PRINTED IN ENGLAND

          _'Tis two score years since CARROLL'S art,
             With topsy-turvy magic,


.
.
.

=== ZEILEN, WORTE, ZEICHEN (8-Bit) ==== 3635  27280 157172 alice-wonderland-pg28885-easy.txt ==== ENDE (20 zeilen) =====

Page 37, "quiet" changed to "quite" (I'm quite tired of)

Page 41, colon changed to period (arm, yer honour.)

Page 42, "wont" changed to "want" (want to stay)

Page 66, closing quotation mark added (to-morrow----")

Page 69, single quotation mark changed to double (cat," said the
Duchess)

Page 91, word "to" added to text (minute or two to)

Page 103, word "as" added to the text (just as she had)

Page 104, "hedge-hog" changed to "hedgehog" (send the hedgehog to)

Page 126, end parenthesis added ("No, never")

Page 153, added an apostrophe (What's in it?)
```

A.2. Werte aus weiteren Läufen

Englischer Text: Alice im Original

Wir vergleichen diese Werte mit denen des englischen Originaltextes

---

mit $L_{max} = 0$, Zeichenkodierung 1-1

Parameter	Wert
b	85
n	5100
c	60

T_{max} =234 ; $m_{T_{\max}} = 234.00$; $s_{T_{\max}} = 0.00$; $m_{T_{\max} \cdot c} = 14040.00$
Übersicht der Fehlerzahlen eines der Läufe:

$\lvert (\tau_t) \rvert$	$e\,(L = 0)$
1	0
10	0
55	0
168	0
224	0
231	0
233	0
234	0
235	1
238	1
252	1
280	3

---

mit $L_{max} = 1$, Zeichenkodierung 1-1

Parameter	Wert
b	85
n	5100
c	60
v_{pad}	2
$L_{\max}$	1

T_{max} =305 ; $m_{T_{\max}} = 305.00$; $s_{T_{\max}} = 0.00$; $m_{T_{\max} \cdot c} = 18300.00$
Übersicht der Fehlerzahlen eines der Läufe:

$\lvert (\tau_t) \rvert$	$e\,(L = 0)$	$e\,(L = 1)$
1	0	
10	0	
55	0	
280	3	0 ($\lvert \left(\tau'_{t'}\right) \rvert = 282$)
298	3	0 ($\lvert \left(\tau'_{t'}\right) \rvert = 300$)
303	3	0 ($\lvert \left(\tau'_{t'}\right) \rvert = 305$)
305	4	0 ($\lvert \left(\tau'_{t'}\right) \rvert = 308$)
306	5	1 ($\lvert \left(\tau'_{t'}\right) \rvert = 310$)
307	5	1 ($\lvert \left(\tau'_{t'}\right) \rvert = 311$)
316	6	1 ($\lvert \left(\tau'_{t'}\right) \rvert = 321$)
351	9	1 ($\lvert \left(\tau'_{t'}\right) \rvert = 359$)
421	20	16 ($\lvert \left(\tau'_{t'}\right) \rvert = 440$)
562	110	145 ($\lvert \left(\tau'_{t'}\right) \rvert = 671$)
843	351	852 ($\lvert \left(\tau'_{t'}\right) \rvert = 1193$)
1405	901	2271 ($\lvert \left(\tau'_{t'}\right) \rvert = 2305$)

Parameter	Wert
b	85
n	5100
c	60
v_{pad}	2
$L_{\max}$	2

mit $L_{max} = 2$, Zeichenkodierung 1-1

T_{max} =363 ; $m_{T_{\max}} = 363.00$; $s_{T_{\max}} = 0.00$; $m_{T_{\max} \cdot c} = 21780.00$
Übersicht der Fehlerzahlen eines der Läufe:

$\lvert(\tau_t)\rvert$	$e\,(L=0)$	$e\,(L=1)$	$e\,(L=2)$
1	0		
10	0		
55	0		
280	3	0 ($\lvert\left(\tau'_{t'}\right)\rvert = 282$)	
351	9	1 ($\lvert\left(\tau'_{t'}\right)\rvert = 359$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 359$)
360	9	5 ($\lvert\left(\tau'_{t'}\right)\rvert = 368$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 372$)
363	9	5 ($\lvert\left(\tau'_{t'}\right)\rvert = 371$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 375$)
364	10	4 ($\lvert\left(\tau'_{t'}\right)\rvert = 373$)	2 ($\lvert\left(\tau''_{t''}\right)\rvert = 376$)
365	10	3 ($\lvert\left(\tau'_{t'}\right)\rvert = 374$)	1 ($\lvert\left(\tau''_{t''}\right)\rvert = 376$)
369	11	4 ($\lvert\left(\tau'_{t'}\right)\rvert = 379$)	1 ($\lvert\left(\tau''_{t''}\right)\rvert = 382$)
386	13	4 ($\lvert\left(\tau'_{t'}\right)\rvert = 398$)	1 ($\lvert\left(\tau''_{t''}\right)\rvert = 401$)
421	20	18 ($\lvert\left(\tau'_{t'}\right)\rvert = 440$)	9 ($\lvert\left(\tau''_{t''}\right)\rvert = 457$)
562	110	130 ($\lvert\left(\tau'_{t'}\right)\rvert = 671$)	260 ($\lvert\left(\tau''_{t''}\right)\rvert = 800$)
843	351	891 ($\lvert\left(\tau'_{t'}\right)\rvert = 1193$)	2036 ($\lvert\left(\tau''_{t''}\right)\rvert = 2083$)
1405	901	2271 ($\lvert\left(\tau'_{t'}\right)\rvert = 2305$)	4575 ($\lvert\left(\tau''_{t''}\right)\rvert = 4575$)

Parameter	Wert
b	7225
n	115600
c	16
v_{pad}	2
$L_{\max}$	1

mit $L_{max} = 1$, Zeichenkodierung 2-1

T_{max} =4910 ; $m_{T_{\max}} = 4910.00$; $s_{T_{\max}} = 0.00$; $m_{T_{\max} \cdot c} = 78560.00$
Übersicht der Fehlerzahlen eines der Läufe:

$\lvert(\tau_t)\rvert$	$e\,(L=0)$	$e\,(L=1)$
1	0	
10	0	
55	0	
280	1	0 ($\lvert\left(\tau'_{t'}\right)\rvert = 280$)
1405	2	0 ($\lvert\left(\tau'_{t'}\right)\rvert = 1406$)
4910	20	0 ($\lvert\left(\tau'_{t'}\right)\rvert = 4929$)

Es ist mit $2-1$-Kodierung bereits bei einem Korrekturlevel $L_{max} = 1$ mit einer kurzen Zeilenlänge von $2 \cdot c = 32$ Zeichen des Originaltextes eine Einspeicherung des Textes möglich – eine **3-2**-Kodierung kann hier für eine kleinere Matrix genutzt werden.

Mechanisch erzeugter Text

Wir betrachten noch das Verhalten des Verfahrens bei einem „künstlichen“ Text: Es handelt sich um *Pfad-/Dateinamen*, geliefert von *find* in einem typischen Benutzerverzeichnis. Als Besonderheit besitzen diese Texte grosse Teile sich wiederholender Zeichenketten, nämlich die gemeinsamen Komponenten des Pfades von allen Dateien in einem bestimmten Verzeichnis an den Zeilenanfängen, s. Seite 90.

Dateinamen, Zeichenkodierung 1-1 Dem entsprechend finden sich mit $L_{max} = 2, c = 60$ nur etwa 5000 einspeicherbare Zeichen (im Gegensatz zu 34500 bei dem natürlichsprachlichen Text).

Parameter	Wert
b	88
n	5280
c	60
v_{pad}	2
$L_{\max}$	2

$T_{\max}$ =84 86 84 77 ; $m_{T_{\max}} = 82.75$; $s_{T_{\max}} = 3.42$; $m_{T_{\max} \cdot c} = 4965.00$
Übersicht der Fehlerzahlen eines der Läufe:

$\lvert(\tau_t)\rvert$	$e\,(L=0)$	$e\,(L=1)$	$e\,(L=2)$
1	0		
10	0		
55	4	1 ($\lvert\left(\tau'_{t'}\right)\rvert = 58$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 58$)
70	7	1 ($\lvert\left(\tau'_{t'}\right)\rvert = 76$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 76$)
74	9	2 ($\lvert\left(\tau'_{t'}\right)\rvert = 82$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 83$)
76	9	0 ($\lvert\left(\tau'_{t'}\right)\rvert = 84$)	
77	9	2 ($\lvert\left(\tau'_{t'}\right)\rvert = 85$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 86$)
78	10	2 ($\lvert\left(\tau'_{t'}\right)\rvert = 87$)	3 ($\lvert\left(\tau''_{t''}\right)\rvert = 88$)
84	12	2 ($\lvert\left(\tau'_{t'}\right)\rvert = 95$)	1 ($\lvert\left(\tau''_{t''}\right)\rvert = 96$)
112	22	15 ($\lvert\left(\tau'_{t'}\right)\rvert = 133$)	10 ($\lvert\left(\tau''_{t''}\right)\rvert = 147$)
168	38	24 ($\lvert\left(\tau'_{t'}\right)\rvert = 205$)	19 ($\lvert\left(\tau''_{t''}\right)\rvert = 228$)
280	86	93 ($\lvert\left(\tau'_{t'}\right)\rvert = 365$)	103 ($\lvert\left(\tau''_{t''}\right)\rvert = 457$)

Dateinamen, mit $L_{max} = 1$, Zeichenkodierung 2-1 (Umkodierung auf ein grösseres Alphabet)

Parameter	Wert
b	7744
n	464640
c	60
v_{pad}	2
$L_{\max}$	1

$T_{\max}$ =539 539 539 539 ; $m_{T_{\max}} = 539.00$; $s_{T_{\max}} = 0.00$; $m_{T_{\max} \cdot c} = 32340.00$
Übersicht der Fehlerzahlen eines der Läufe:

$\lvert(\tau_t)\rvert$	$e\,(L=0)$	$e\,(L=1)$
1	0	
10	0	
55	1	0 ($\lvert\left(\tau'_{t'}\right)\rvert = 55$)
280	2	0 ($\lvert\left(\tau'_{t'}\right)\rvert = 281$)
421	3	0 ($\lvert\left(\tau'_{t'}\right)\rvert = 423$)
492	3	0 ($\lvert\left(\tau'_{t'}\right)\rvert = 494$)
527	6	0 ($\lvert\left(\tau'_{t'}\right)\rvert = 532$)
536	7	0 ($\lvert\left(\tau'_{t'}\right)\rvert = 542$)
539	7	0 ($\lvert\left(\tau'_{t'}\right)\rvert = 545$)
540	7	1 ($\lvert\left(\tau'_{t'}\right)\rvert = 546$)
541	7	1 ($\lvert\left(\tau'_{t'}\right)\rvert = 547$)
545	7	1 ($\lvert\left(\tau'_{t'}\right)\rvert = 551$)
562	7	1 ($\lvert\left(\tau'_{t'}\right)\rvert = 568$)
843	21	1 ($\lvert\left(\tau'_{t'}\right)\rvert = 863$)
1405	23	2 ($\lvert\left(\tau'_{t'}\right)\rvert = 1353$)

Dateinamen, mit $L_{max} = 2$, Zeichenkodierung 2-1 Die Umkodierung auf ein grösseres Alphabet erreicht hier das Ziel der Einspeicherung von 159903 Zeichen dann mit $L_{max} = 2$ (und zwar schon mit $c = 40$, vorletztes Protokoll).

Parameter	Wert
b	7744
n	278784
c	36
v_{pad}	2
$L_{\max}$	2

$T_{\max}$ =1007 1007 1007 1007 ;
$m_{T_{\max}} = 1007.00$; $s_{T_{\max}} = 0.00$; $m_{T_{\max} \cdot c} = 36252.00$
Übersicht der Fehlerzahlen eines der Läufe:

$\lvert(\tau_t)\rvert$	$e\,(L=0)$	$e\,(L=1)$	$e\,(L=2)$
1	0		
10	0		
55	0		
280	1	0 ($\lvert\left(\tau'_{t'}\right)\rvert = 280$)	
843	4	1 ($\lvert\left(\tau'_{t'}\right)\rvert = 846$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 846$)
984	9	1 ($\lvert\left(\tau'_{t'}\right)\rvert = 992$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 992$)
1002	11	1 ($\lvert\left(\tau'_{t'}\right)\rvert = 1012$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 1012$)
1007	11	1 ($\lvert\left(\tau'_{t'}\right)\rvert = 1017$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 1017$)
1008	13	2 ($\lvert\left(\tau'_{t'}\right)\rvert = 1020$)	1 ($\lvert\left(\tau''_{t''}\right)\rvert = 1021$)
1009	13	2 ($\lvert\left(\tau'_{t'}\right)\rvert = 1021$)	1 ($\lvert\left(\tau''_{t''}\right)\rvert = 1022$)
1011	13	2 ($\lvert\left(\tau'_{t'}\right)\rvert = 1023$)	1 ($\lvert\left(\tau''_{t''}\right)\rvert = 1024$)
1019	13	3 ($\lvert\left(\tau'_{t'}\right)\rvert = 1031$)	1 ($\lvert\left(\tau''_{t''}\right)\rvert = 1033$)
1054	18	3 ($\lvert\left(\tau'_{t'}\right)\rvert = 1071$)	1 ($\lvert\left(\tau''_{t''}\right)\rvert = 1073$)
1124	18	3 ($\lvert\left(\tau'_{t'}\right)\rvert = 1141$)	1 ($\lvert\left(\tau''_{t''}\right)\rvert = 1143$)
1405	21	3 ($\lvert\left(\tau'_{t'}\right)\rvert = 1425$)	1 ($\lvert\left(\tau''_{t''}\right)\rvert = 1427$)

Parameter	Wert
b	7744
n	309760
c	40
v_{pad}	2
$L_{\max}$	2

$T_{\max}$ =1997 1997 1997 1997 ;
$m_{T_{\max}} = 1997.00$; $s_{T_{\max}} = 0.00$; $m_{T_{\max} \cdot c} = 79880.00$
Übersicht der Fehlerzahlen eines der Läufe:

$\lvert(\tau_t)\rvert$	$e\,(L=0)$	$e\,(L=1)$	$e\,(L=2)$
1	0		
10	0		
55	0		
280	6	1 ($\lvert\left(\tau'_{t'}\right)\rvert = 285$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 285$)
1405	35	3 ($\lvert\left(\tau'_{t'}\right)\rvert = 1439$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 1441$)
1997	88	4 ($\lvert\left(\tau'_{t'}\right)\rvert = 2084$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 2087$)

Parameter	Wert
b	7744
n	464640
c	60
v_{pad}	2
$L_{\max}$	2

$T_{\max}$ =1331 1331 1331 1331 ;
$m_{T_{\max}} = 1331.00$; $s_{T_{\max}} = 0.00$; $m_{T_{\max} \cdot c} = 79860.00$
Übersicht der Fehlerzahlen eines der Läufe:

$\lvert(\tau_t)\rvert$	$e\,(L=0)$	$e\,(L=1)$	$e\,(L=2)$
1	0		
10	0		
55	1	0 ($\lvert\left(\tau'_{t'}\right)\rvert = 55$)	
280	2	0 ($\lvert\left(\tau'_{t'}\right)\rvert = 281$)	
1331	23	2 ($\lvert\left(\tau'_{t'}\right)\rvert = 1353$)	0 ($\lvert\left(\tau''_{t''}\right)\rvert = 1354$)

B. Verwendete Programme

Neben Programmen für besondere Aufgabenstellungen, welche Unabhängig von dieser Arbeit zum Teil für Seminar- und Projektarbeiten, einer Masterarbeit [(1)] verwendet und z.T. auch zur interaktiven Demonstration bei einer Ausstellung eingesetzt wurden, ergabe sich die Notwendigkeit, zur automatisierten Versuchsdurchführung bei der Erforschung der Fragestellungen der Arbeit, ein flexibel konfigurierbares System zur Ablaufgestaltung und Datensatzkodierung/-dekodierung zu entwickeln.

B.1. Einzelne Programme zur Untersuchung von Eigenschaften von Matrixspeichern und Kodierungen

- gehirn.py – zur Berechnung von Informationsgehalten und Überprüfung der Fehlerlosigkeit von Speichern (enthalten in [4]). Basierend darauf:
- igehirn.py – erste Untersuchungen des Einspeicherns von Sequenzen und interaktives Experimentieren mit relativ kleinen Matrixspeichern [(2)] [4]
- mneu.py – zur systematischen Prüfung der Erkennungsleistung selektiver Kodierung zum Matching
- vidasemu – anfangs dem Vidas-System nachempfundener Simulator für Simulation des sogenannten *Robot-Modells*, welches mit verschiedenen *Kodierfeldern* Merkmale aus dem *Merkmalswald* extrahiert und mittels Assoziativmatrizen verarbeitet. Extrahiert Merkmale aus 2-dimensionalen Gittern von Zeichen.

[(1)] und in Ergaenzung des Vidas-Systems auch zu Experimenten in der Vorlessung „Assoziative Programmierung“. Zu Vidas (Simulationsumgebung für die Programmierung von Assoziativmaschinen in unterschiedlichen Kontexten) siehe [Bentz, Dierks 12]

[(2)] das „i“ steht dabei sowohl für *Interaktion* wie auch im üblichen Sinne als Bezeichnung eines *Index* wie in a_i.

B.2. igehirn.py

Erläuterung der Bildschirmdarstellung wie sichtbar in Abb. 3.2 und folgende:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Programm zum Lernen von Sequenzen in Assoziativmatrizen V 0.52
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
contact: rosenschein@cs.uni-hildesheim.de, matrizen@felderundfiguren.de

Installation:
==================================

Benoetig werden:

python  (fuer win user: latest 2.7 version von python.org, 2.6 tut auch),
Tkinter (bereits darin enthalten),

PIL     (google:"python pil" findet passenden
download link, version beachten)

[...]
Konfiguration der Matrixgroesse:
==================================
[...]

Bildschirmdarstellung:
==================================

Lernvektoren:
----------------------------------

Die In-/Out-Vektoren sind senkrecht aufgetragen. Dabei sind nur die
'Knoepfe' gemeint. (Beschreibung der *kleinen* AnzeigePunkte immer rechts
von den Knoepfen: Siehe unten 'Outputs')

Belegung der Matrix:
----------------------------------

Ganz rechts wird der Inhalt des (immer quadratischen Zeilen x Zeilen)
Matrixspeichers dargestellt
```

```
Automatischer Lern-/Pruef-Lauf:
-------------------------------

1) Lernen
-------------------------------

Das Programm speichert nach jeder Aenderung (immer wieder in eine leere
Matrix):

Von links nach rechts werden die Spalten 0,1,...,i,i+1,...,n2
eingespeichert:

lerne Input:Spalte_0 -> Output:Spalte_1
lerne Input:Spalte_1 -> Output:Spalte_2
lerne Input:Spalte_2 -> Output:Spalte_3
.
.

Bis auf den Vektor ganz links sind also alle einmal Outputs, bis auf den
Vektor ganz rechts sind alle einmal Inputs. Jeder Vektor lernt seinen
rechten Nachbarn, mit Ueberlappung.

(Will man also einmal fuer andere Zwecke als Sequenz-Untersuchungen nur
Einzelpaare lernen, so muss zwischen diesen immer eine Spalte freigelassen
werden. Das Programm 'versucht' zwar, auch diese zu lernen. Aber die
Lernregel bewirkt bekanntlich, dass wenn In- oder Out-Vektor komplett 0-en
sind, in der Speichermatrix keine Bits gesetzt werden)

2) Anfragen
-------------------------------

Anschliessend wird jeder dargestellte Vektor am Matrixspeicher abgefragt.
Das Ergebnis wird in den kleinen Punkten jeweils direkt rechts neben jedem
Vektor dargestellt. (Auch der Vektor ganz rechts wird angefragt)

3) Pruefung
-------------------------------

Zuletzt wird fuer jeden Output geprueft, ob dieser mit dem Soll-Output (das
ist direkt rechts daneben ja der naechste Vektor) uebereinstimmt, und bei

Uebereinstimmung:   Output+Zielvektor werden GRUEN gefaerbt
-----

Nicht-Uebereinstimmung:   Output+Zielvektor werden ROT   gefaerbt
---

!!! Fuer den ganz rechten Vektor gilt der Vektor ganz links als
Kontrollvektor, obwohl dieser nicht also solcher dazu gelernt wurde! Das
ist also oft unsinnig, im Regelfalle wird man diesen ganz rechten Vektor
also nicht verwenden. Hm...  (*)
```

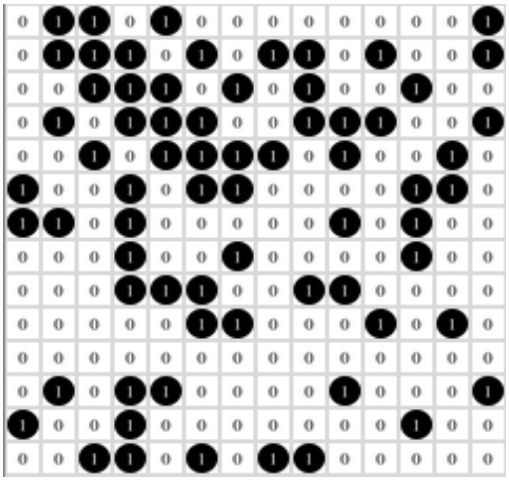
```
Zusaetzliche Markierungen:
----------------------------------

Doublettenpruefung
----------------------------------

Jeder Vektor, der mehrfach auftritt,              wird GELB   markiert
----
(Das ROT der eventuellen nicht-Uebereinstimmung des Zielvektors wird
hierdurch falls vorhanden ueberschrieben; der Fehlerzustand ist aber
noch im ROT des Outputvektors zu erkennen)
[...]

(*) ich benutze das gelegentlich, um an dieser
Stelle alles vorbeizurollen, ob irgendwo dabei was
rotes erscheint. Dadurch erkennt man, ob bestimmte
Beziehungen zu lernen, ueberfluessig ist. (Da wird
ja nicht gelernt, nur geprueft.)

Tastenbelegungen:
==================================
[...]

Dateinamen / Dateiformate:
==================================
[...]
```

Unter [4] finden sich die vollständige Beschreibung sowie das Programm.

B.3. Simulationssystem für verbundene Assoziativspeicher

Dieses System noch ohne Namen realisiert die datengesteuerte Verschaltung von *mehreren Assoziativspeichern* und Einsatz unterschiedlicher Kodierer.
Der Gesamtaufbau eines Experiments wird in Konfigurationsdateien festgelegt, welche auch automatisierte Benutzerinteraktionen beschreiben können, die auf dem Instanziierten Modell ablaufen sollen.
Die Verwendung des Systems erfolgt nach Bedarf gesteuert über Skript-Dateien (zum Beispiel zum Generieren einiger der hier wiedergegebenen Bilder) des jeweils verwendeten Betriebssystems oder interaktiv über ein Web-Interface.

B.4. Speichermatrix zu *langer Sequenz in kleiner Matrix*



C. Glossar

Ohne Anspruch auf Vollständigkeit sind hier neben Abkürzungen und Symbolen einige Begriffe gelistet, welche in besonderer Weise gebraucht werden.

$\oplus$ Gemeinsames Auftreten in **einem** Vektor, ähnlich **oder**

Adaptivität Beispiel: Die Daten selbst bestimmen ihre Zerlegung in verschieden grosse Blöcke

AM Assoziativmaschine, Assoziativmatrix

Assoziation τ Tupel von (Frage,Antwort). Menge von Einzelassoziationen

Assoziationskette Überlappende Sequenz von n Stück (F^i, A^i)-Tupeln aus den direkten Nachbarn eines $n+1$-elementigen $(V^0, \ldots, V^n)$

Assoziativmatrix, Assoziativspeicher, Matrixspeicher binärwertige Matrix (a_{ij})

Assoziieren (eine Frage an eine Assoziativmatrix stellen) Durchführen des Abfrageprozesses (binärlogische Matrixmultiplikation mit nachfolgender Schwellwertreduktion) an einem Assoziativspeicher

Aufteilen und Auffüllen Ein Verfahren des „reblocking“

Einzelassoziationen Gemeinsames Auftreten in **zwei** Vektoren, entspricht **und**

fortgesetztes Assoziieren Die Ausgabe des Matrixspeichers wieder als Eingabe verwenden

Indexfolge eine geordnete Menge von Merkmalsvektoren, welche als Assoziationskette in einem Matrixspeicher hinterlegt ist. Durch n-maliges fortgesetztes Assoziieren von einem Startwert aus kann jedem n eineindeutig ein Merkmalsvektor zugeordnet werden.

KNN künstliches neuronales Netz

Lernregel Vorschrift zum Einschreiben eines Vektors in einen Assoziativspeicher

Merkmalsvektor Menge von Merkmalen

p_V Belegungsdichte eines Vektors

$p_{(a_{ij})}, P$ Belegungsdichte einer Matrix

„reblocking“ Neuaufteilung der Datensätze während des Einspeicherungsprozesses

Schwellwertreduktion Merkmale nicht-maximaler Erregung löschen

T Anzahl der Vektoren (Text)

$|\,(\tau_t)\,|$ Anzahl der Assoziationen;
im Falle von Sequenzbildung (wenn kein „reblocking“): $= T - 1$

triviale Merkmalsvektoren tragen keine strukturierte Information;
also (0) und (1)

Literaturverzeichnis

[Ackermann 00] M. Ackermann (2000) *Statistische Korpusanalyse zum Extrahieren von semantischen Wortrelationen*, Dissertation; Hildesheim: Universität Hildesheim

[Bentz et al. 89] Hans-J. Bentz, Michael Hagström, Günther Palm (1989) *„Information Storage and Effective Data Retrieval in Sparse Matrices“* in: „Neural Networks“, Vol. 2, S. 289-293, Maxwell Pergamon Macmillan

[Bentz 06] Hans-Joachim Bentz (2006) *„Die Suchmaschine SENTRAX. Grundlagen und Anwendungen dieser Neuentwicklung“* in: „Proceedings des Fünften Hildesheimer Evaluierungs- und Retrievalworkshops“, Hildesheim

[Bentz, Dierks 12] H.-J. Bentz und A. Dierks (2012) *Neuromathematik und Assoziativmaschinen* unveröffentlichtes Buchmanuskript; eingereicht bei Springer-Verlag

[Carroll 69] Lewis Carroll, *Alice's Abenteuer im Wunderland* Aus dem Englischen von Antonie Zimmermann, Leipzig Johann Friedrich Hartknoch. Originally published in 1869. [2]

[Dierks 05] A. Dierks (2005) *VidAs - Aufbau einer robusten, frei programmierbaren Maschine aus Assoziativmatrizen. Simulation und Hardware-Lösung*, Dissertation; Hildesheim: Universität Hildesheim

[Fay 02] R. Fay (2002) *Hierarchische neuronale Netze zur Klassifikation von 3D-Objekten*, Master's thesis, University of Ulm, Department of Neural Information Processing

[Frobese 09] D. Frobese (2009) *E-Mail-Kategorisierung und Spam-Detektion mit SENTRAX*, Universität Hildesheim 2009, Dissertation; zugl. Berlin: Franzbecker

[Heitland 94] M. Heitland (1994) *Einsatz der SpaCAM-Technik für ausgewählte Grundaufgaben der Informatik*, Dissertationsschrift, Hildesheim

[Hagström 96] M. Hagström (1996) *Textrecherche in großen Datenmengen auf der Basis spärlich codierter Assoziativmatrizen*, Dissertationsschrift, Hildesheim

[Hawkins et al. 11] J. Hawkins et al. (2011) *Hierarchical Temporal Memory*, von https://www.numenta.com/htm-overview/education/HTM_CorticalLearningAlgorithms.pdf, Version 0.2.1 vom 12.11.2011

[Janotta 11] J.M. Janotta (2011) *Untersuchung von Eigenschaften assoziativer Codierungen für spezielle Wortgruppen*, Masterarbeit; Hildesheim: Universität Hildesheim

[Kopold 97] W. Kopold (1997) *Zusammensetzbarkeit und Ähnlichkeitserhaltung bei spärlichen Codierungen*, Dissertation; Ulm: Universität Ulm

[Luba 01] T. Luba (2001) *Konkordanzen in langen Zeichenketten*, Dissertation; Hildesheimer Informatik-Berichte 1/2001; Hildesheim: Universität Hildesheim

[Sommer, Palm 99] F.T. Sommer, G. Palm (1999) „Improved bidirectional retrieval of sparse patterns stored by Hebbian learning“, in „Neural Networks“ 12, S. 281-297

[Na nhongkai 06] Suriya Na nhongkai (2006) *Untersuchungen zur sprachübergreifenden, bilingualen Suche mit Hilfe der Konzeptnetz-Technologie der SENTRAX-Engine*, Dissertation an der Universität Hildesheim

[Palm 80] G. Palm (1980) *„On Associative Memory“*, in *Biological Cybernetics Nr. 36, S. 19-31*, Springer-Verlag

[Palm 82] G. Palm (1982) *Neural Assemblies. An Alternative Approach to Artificial Intelligence*, Springer, Berlin, Heidelberg, New York

[Palm 88] G. Palm (1988) *„Assoziatives Gedächtnis und Gehirntheorie“* in *Spektrum der Wissenschaft Nr. 6/1988, S. 54-64*, Spektrum der Wissenschaft Verlag, Heidelberg

[Palm et al. 10] A. Knoblauch, G. Palm, F.T. Sommer (2010) „Memory Capacities for Synapic and Structural Plasticity“, in „Neural Computation“ Vol. 22 No. 2, S. 289-341, MIT Press Cambridge

[Palm 13] G. Palm (2013) „Neural associative memories and sparse coding“, in: „Neural Networks“ Vol. 37, S. 165-171

[Potter 92] J. Potter (1992) *Associative computing: a programming paradigm for massively parallel computers*, New York: Plenum Press

[Rojas 93] R. Rojas (1993) *Theorie der neuronalen Netze*, Springer-Verlag

[Töpfer 11] T. Töpfer (2011) *Matrixspeicher und Informationskapazität*, Masterarbeit; Hildesheim: Universität Hildesheim

Websites:

[1] A. Dierks *QUINTAS - Ein durch eine Assoziativmaschine gesteuertes Vehikel*,
http://assoziativmaschine.de
am 10.4.2012

[2] *Alice's Abenteuer im Wunderland* bei „Project Gutenberg“,
http://www.gutenberg.org/1/9/7/7/19778/
am 22.10.2012

Programme:

[3] F. Rosenschein *Programm zu Tupeluntersuchugen 2010*,
http://felderundfiguren.de/programme/tupeler

[4] F. Rosenschein *Programm zu Sequenzprüfungen 2011*,
http://felderundfiguren.de/programme/igehirn

[5] Programm *„SENTRAX“*, 2004
http://sentrax.de/praesentationen/gettingstarted/seiten/1.html
am 1.5.2012

Bildquellen:

[6] *Ausschnitt aus der Reproduktion eines Filmplakat der* $M \cdot G \cdot M$, eigene Aufnahme

Abbildungsverzeichnis

Tabellenverzeichnis